This chapter describes the AOCE Standard Catalog Package. The AOCE Standard Catalog Package provides a high-level interface that makes it easy for you to add catalog-browsing, authentication, and alias-resolution services to your applications.

This chapter assumes you are familiar with the nature and use of AOCE catalogs and authentication services.

If you want to design and implement your own interface to AOCE catalogs, see the chapter "Catalog Manager" in this book.

About the Standard Catalog Package

The AOCE Standard Catalog Package provides a high-level interface to the AOCE Catalog Manager. It makes it easy for you to add catalog-browsing and record-selection services to your application.

The Standard Catalog Package provides functions that

- Display a dialog box that allows users to enter their password for their authentication identity.
- Display and return information from a Catalog-Browsing panel that you can place in your window. The Catalog-Browsing panel lets users browse catalogs and select records.
- Display and return information from a Find panel that you can place in your window. The Find panel lets users search catalogs for a record if they know all or part of the record's name.
- Resolve aliases of records, catalogs, and other objects in the AOCE catalog system.
- Return icons for records, catalogs, and other AOCE components.
- Return lists of categories of catalog items available on a system (such as printers or users) and of types of items in these categories (such as LaserWriter and ImageWriter printers).

The Standard Catalog Package is a client of the AOCE Catalog and Authentication Managers. You do not have to call the underlying AOCE services directly to add catalog services to your application.

Finding and Selecting Records

The Standard Catalog package provides two standard interfaces for finding and selecting records in catalogs: a Catalog-Browsing panel and a Find panel. Catalog panels are described in "Creating, Displaying, and Disposing of a Catalog-Browsing Panel," beginning on page 4-29, and "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51. Find panels are described in "Creating, Displaying, and Disposing of a Find Panel," beginning on page 4-61, and "Handling Events in a Find Panel," beginning on page 4-75. You can place a panel in any window you wish and provide menu items and dialog-box controls that make it easier for the user to browse or search for catalogs.

Figure 4-1 shows a Catalog-Browsing panel in use in an application's window, and Figure 4-2 shows a Find panel as it appears in a window of the AppleMail application. It is important to note that the use of such panels is not restricted to looking up addresses; you can use these interfaces to allow users to browse and search AOCE catalogs no matter what their content. For example, if you use an AOCE catalog to store information about routing forms (who should sign the form, the sequence in which to route it, and so forth), you can use the panels to help the user fill in and route a particular form.

Figure 4-1 A Catalog-Browsing panel in an application window

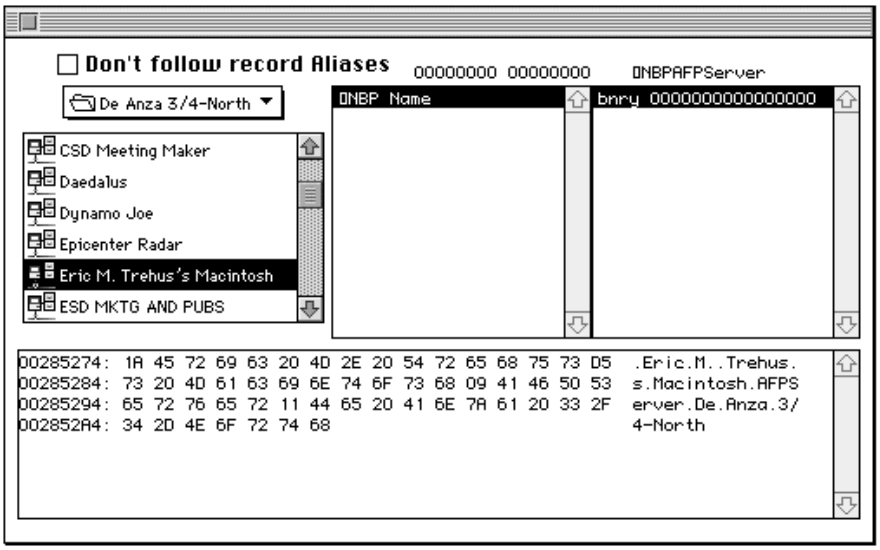
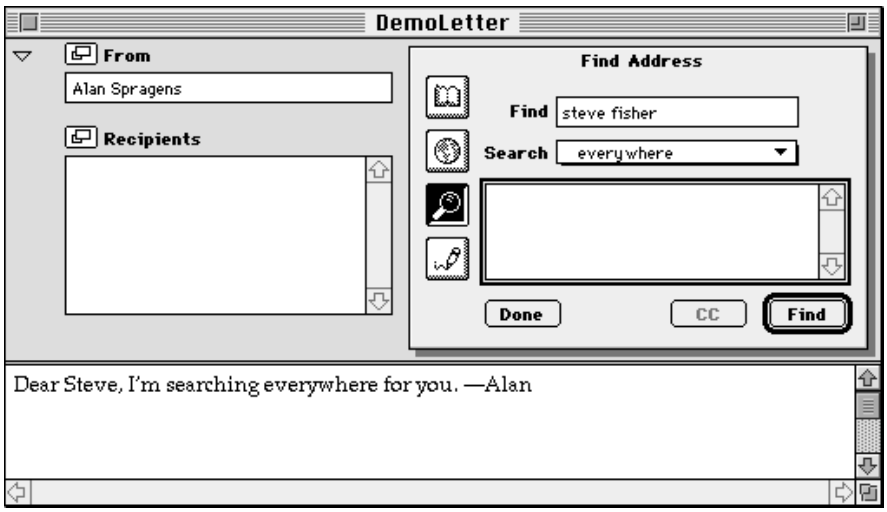


Figure 4-2 A Find panel in an application window



Using the Standard Catalog Package

This section describes how to use Standard Catalog Package routines to determine if AOCE is available, to authenticate a user, to create a Catalog-Browsing panel, and to handle Catalog-Browsing panel events. It also describes how to create and dispose of a Find panel.

Testing for the Presence of the Standard Catalog Package

Before using Standard Catalog Package functions in your application, you must use the Gestalt Manager to ensure that the system on which your application is running supports AOCE and the Standard Catalog Package.

To verify that the AOCE Collaboration toolbox is available, call the `Gestalt` function with the selector `gestaltOCEToolboxAttr`. If the Collaboration toolbox is present but not running (for example, if the user deactivated it from the PowerTalk Setup control panel), the `Gestalt` function sets the bit `gestaltOCETBPresent` in the `response` parameter. If the Collaboration toolbox is running and available, the function sets the bit `gestaltOCETBAvailable` in the `response` parameter. The Gestalt Manager is described in the chapter "Gestalt Manager" of *Inside Macintosh: Operating System Utilities*.

To determine the version of the Standard Catalog Package that is available, call the `Gestalt` function with the selector `gestaltSDPStandardDirectoryVersion`. The function returns the version number of the Standard Catalog Package in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Standard Catalog Package is not present and available, the `Gestalt` function returns 0 for the version number. Similarly, to determine the version of the Find panel, use the selector `gestaltSDPFindVersion`. To determine the version of the prompt-for-identity dialog box, use the selector `gestaltSDPPromptVersion`.

Listing 4-1 shows a function that checks for the availability of AOCE routines and returns `true` only if the Standard Catalog Package is installed and available.

Listing 4-1 Testing for the Standard Catalog Package

```
Boolean MyTestForStandardCatalog(void)
{
   OSErr    err;
   long     response;

   err = Gestalt(gestaltSDPStandardDirectoryVersion, &response);
   if ((err != noErr) || (response == 0))
      return false;
```

```
    return true;
}
```

Creating an Authentication Identity

The PowerTalk Key Chain lets the owner or principal user of a Macintosh computer enter a name, password, and other identifying information for PowerShare servers and whatever access modules are installed. The PowerTalk software protects that information with a single password, referred to as the ***Key Chain Access Code*** . The Authentication Manager takes the name and password set in the Key Chain and issues the computer a ***local identity***. The AOCE toolbox can then gain access to the PowerShare server and access module services without requiring the user to log on again or enter another password.

Before the first time you send a message or let the user browse catalogs, you must provide identification to prove that the caller is an authorized user of the system. The `SDPPromptForID` function provides two versions of a dialog box that allows the user to identify himself or herself as one of the authorized users of the system (Figure 4-3). The simpler version of this dialog box allows the user to enter only the Key Chain Access Code. The function then returns the local identity to your application. The other version of this dialog box (also shown in Figure 4-3) allows the user to select a catalog and enter a name and password. The name must correspond to a record in the selected catalog. The function then verifies that the user has entered the correct access code and returns a ***specific identity***. If the user logs on as a guest, the function returns 0 for the identity.

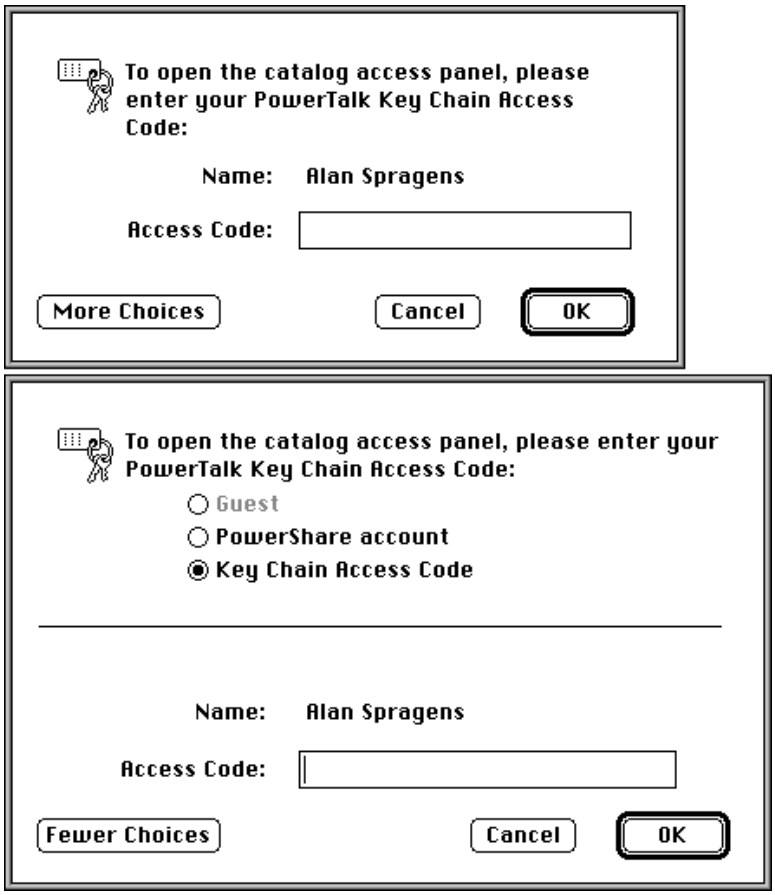
The "More Choices" version of the dialog box provides radio buttons corresponding to the three possibilities: Guest, PowerShare account (specific identity), or Key Chain Access Code (local identity). You can specify which of these options are enabled. You can also specify a catalog or a user or group record to be displayed initially in the dialog box, and you can restrict the user to this initially displayed selection.

You must provide an identity when you call many of the Standard Catalog Package functions and most of the functions in the Catalog and Authentication Managers.

Note
Local AOCE resources such as personal messaging service access modules (MSAMs) are not available to a user who logs on with a specific identity or as a guest. ◆

If the user has never set up a local identity or selected any AOCE catalogs, the `SDPPromptForID` function prompts the user to make the appropriate choices.

Figure 4-3 Authentication dialog box



Before using Standard Catalog Package services, your application must determine the user's authentication identity. To discover if the user has previously been authenticated, without prompting the user, call the `AuthGetLocalIdentity` function, described in the "Authentication Manager" chapter of this book. If that function returns an error, then call the `SDPPromptForID` function to allow the user to unlock the local identity, set one up if none exists, or log on as a specific identity or guest.

The routine shown in Listing 4-2 first checks for an existing local identity. If none has been set up (if, for example, the system has just been started) or if the PowerTalk Key Chain is locked, the routine calls `SDPPromptForID`. The resulting authentication identity is stored by the application in the `AuthIdentity` pointer parameter `userIdentity`. If the local identity is already set up, it is returned in the authentication parameter block (type `AuthParamBlock`).

Listing 4-2 Getting an authentication identity

```
OSErr MyGetUserIdentity(AuthIdentity *userIdentity)
{
   SDPIdentityKind   selectedKind;
   AuthParamBlock    theAuthParamBlock;
   OSErr             status;

   status = AuthGetLocalIdentity(&theAuthParamBlock, FALSE);
   if (status == noErr) { /* Local identity is already set up. */
      *userIdentity =
         theAuthParamBlock.getLocalIdentityPB.theLocalIdentity;
   }
   else if (status == kOCELocalAuthenticationFail) {
      status = SDPPromptForID(
               userIdentity,  /* AuthIdentity *id */
               NULL,          /* default guest prompt */
               NULL,          /* default specific prompt */
               NULL,          /* default local prompt */
               NULL,          /* RString *recordType */
               (              /* SDPIdentityKind permittedKinds */
                  kSDPLocalIdentityMask | /* local identity or */
                  kSDPSpecificIdentityMask /* specific identity */
               ),
               &selectedKind, /* SDPIdentityKind *selectedKind */
               NULL,          /* RecordID *loginFilter */
               0              /* SDPLoginFilterKind filterKind */
            );
   }
   else {
      /* could check for (status == kOCESetupRequired) */
   }
   return status;
}
```

Creating a Catalog-Browsing Panel

This section illustrates how to create a Catalog-Browsing panel. The Catalog-Browsing panel provides a scrolling list and pop-up menu that you can place in any window. This list shows AOCE catalogs and their contained nodes and records. It also shows hierarchical file system (HFS) volumes, catalogs, and files so that the user can find AOCE information cards, aliases, and personal catalogs. The panel allows users to examine the contents of AOCE catalogs and to select a record in a catalog. Figure 4-4 shows a Catalog-Browsing panel, and Figure 4-1 on page 4-4 shows a panel in use in an

application window. Note that the font, size, and style of the characters in the Catalog-Browsing panel are those you set for the window before you call a routine to create a panel.

Figure 4-4 A Catalog-Browsing panel

In addition to the Catalog-Browsing panel, you can display a Personal-Catalog panel. This panel is identical to the Catalog-Browsing panel except that it displays only the contents of the user's default personal catalog. The Personal-Catalog panel looks just like the Catalog-Browsing panel in Figure 4-4 except that it does not include a pop-up menu.

Listing 4-3 illustrates the use of the `SDPNewPanel` function to create a new panel.

Listing 4-3 Using the `SDPNewPanel` function to create a new panel

```
OSErr MyCreateNewPanel(SDPPanelHandle *newPanel,
                       WindowPtr      theWindow,
                       Rect           *bounds,
                       AuthIdentity   identity)
{
   PackedRStringListHandle types = NULL;
   unsigned short          typeCount;
   RStringPtr              *typeList = NULL;
   DirEnumChoices          enumFlags;
   DirMatchWith            matchTypeHow;
   RStringPtr              *myCategory;
   Boolean                 typesIsResource = false;
   OSErr                   result;

   matchTypeHow = kExactMatch;
   enumFlags = kEnumDistinguishedNameMask + kEnumAliasMask + kEnumDNodeMask;

   myCategory = (RStringPtr *)GetResource('rstr',kMyCategory);
   result = !myCategory?resNotFound:ResError();

   if(result)
   {
```

```
      goto UseDefaultTypeList;
   }
   if(!result)
   {
      result = SDPGetCategoryTypes(*myCategory ,&types);
      if(result)
      {
         /* Get default type list from a resource. */
         UseDefaultTypeList:

         types =(PackedRStringListHandle)GetResource('rtyp',kTypeListID);
         result = !types?resNotFound:ResError();
         DisposHandle((Handle)myCategory);
         typesIsResource = true;
      }
   }

   if(!result)
   {
      typeCount = OCEDNodeNameCount(*types);
      typeCount++; /* Make space for DNode record type. */
      typeList = NewPtr(typeCount * sizeof(RStringPtr));
      OCEUnpackPathName(*types,typeList,typeCount);

      /* Allow aliases to DNodes to be shown by adding the following type.*/
      typeList[typeCount-1] = OCEGetIndRecordType(kDNodeRecTypeNum);
   }

   if(!result)
   {
      result = SDPNewPanel(&newPanel,
                theWindow,
                &bounds,
                true,
                true,
                NULL,
                typeList,
                typeCount,
                identity,
                enumFlags,
                matchTypeHow,
                0);
   }
```

```
    if(typesIsResource)
    {
       ReleaseResource((Handle),types);
    }
    else
    {
       DisposHandle((Handle),types);
    }
    DisposPtr(typeList);
    return(result);
}
```

Handling Catalog-Browsing Panel Events

When you receive a window event in your application that contains a Catalog-Browsing panel, you must first determine whether it took place in the panel. For mouse-down events, you can check the coordinates of the event to see if they are in the panel. You must keep track of where the user is working to know how to handle key-down events. For example, you can draw a heavy border (called a ***focus rectangle*** or focus box) around the panel or around the content portion of the window, according to the last location of a mouse-down event. The focus rectangle indicates to the user that the area it encloses is active and that any subsequent key-down event pertains to that portion of the window.

If the event took place in the Catalog-Browsing panel, you should call the `SDPPanelEvent` function, passing in the event record. If the return value of `SDPPanelEvent` tells you the user has double-clicked a record displayed in the panel or otherwise changed the selection, call `SDPGetPanelSelectionSize` and `SDPGetPanelSelection`.

If the return value of `SDPPanelEvent` tells you the event is an update event, call `SDPUpdatePanel`. If the return value says that the user has selected an item other than a record, and your application presents a way to open it, such as a menu command, you can open the item by calling `SDPOpenSelectedItem`. You can determine what the user selected in the panel by calling `SDPGetPanelSelectionState`. Finally, you should adjust your application menus appropriately to reflect their status in the wake of the event.

The routines shown in Listing 4-4 illustrate event handling in a Catalog-Browsing panel. The first application-defined function, `MyProcessEvent`, receives events from the application's main event loop and dispatches them, according to their type, to subroutines also shown in the listing.

Listing 4-4 omits some utility routines that perform functions such as `MyGetPanelForThisWindow`, which takes a window pointer and the address of a panel handle as parameters and returns a Boolean value. It returns `true` if it successfully sets the panel handle to the Catalog-Browsing panel associated with the window. This

application associates a panel with a window by storing the panel handle in the window record's reference constant. The application-defined function `MyDisplayDataForSelection` shows code that retrieves a catalog record selected by the user and unpacks the record information. The listing omits code that extracts other information and that displays the unpacked information.

Listing 4-4 Handling events in a Catalog-Browsing panel

```
void MyProcessEvent(EventRecord *ev)
{
   SDPPanelHandle panel;

   switch (ev->what) {
         case mouseDown:
         case mouseUp:
            MyHandleMouseUp(ev);
            break;
         case keyDown:
         case autoKey:
            if (MyGetPanelForThisWindow(FrontWindow(),&panel) == true)
               MyHandleKeyDownsForPanel(FrontWindow(),ev,panel);
            break;
         case updateEvt:
            MyHandleUpdates((WindowPtr)ev->message);
            break;
         case activateEvt:
            MyHandleActivates(ev);
            break;
         case osEvt:
            MyHandleSREvt(ev->message);
            break;
         case nullEvent:
            MyHandleIdle(ev,FrontWindow());
            break;
   }
}

void MyHandleMouseUp(EventRecord *ev)
{
   WindowPtr window;
   Rect limit;
   SDPPanelState whatHappened;
   SDPPanelHandle panel;
```

```
   OSErr err;

   SetRect(&limit,-32000,-32000,32000,32000);
   switch (FindWindow(ev->where,&window))
   {
      case inDrag:
         DragWindow(window,ev->where,&limit);
         break;
      case inGoAway:
         if (TrackGoAway(window,ev->where))
            gDone = true;
         break;
      case inMenuBar:
         /* MyDoMenuCommand executes the menu command the user chose. */
         MyDoMenuCommand(FrontWindow(),MenuSelect(ev->where));
         break;
      case inSysWindow:
         SystemClick(ev,window);
         break;
      case inContent:
         if (MyGetPanelForThisWindow(window,&panel) == true)
         {
            /* MyClickedInOurPanel returns true if click was in panel. */
            if (MyClickedInOurPanel(ev, (*panel)->bounds) == true)
            {
                  /* Pass the event to the Standard Catalog Package. */
                  err = SDPPanelEvent(panel, ev, &whatHappened);
                  if (err == noErr)
                     MyHandleUserSelection(window,whatHappened,panel);
            }
         }
         break;
   }
}

void MyHandleKeyDownsForPanel(WindowPtr window, EventRecord *ev,
                              SDPPanelHandle panel)
{
   short theChar;
   SDPPanelState whatHappened;
   OSErr err;
   /* dummy string that holds keys pressed by the user */
   RString sKeyDowns = {smRoman,1,{'?'}};
```

```
   theChar = ev->message & charCodeMask;
   if ((ev->modifiers & cmdKey) != 0)
      MyDoMenuCommand(window, MenuKey(theChar));
   else
   {
      /* Did they press the Return key? If so,
         open the currently selected item. */
      if (theChar == 0x0D)
      {
         err = SDPOpenSelectedItem(panel,&whatHappened);
         MyHandleUserSelection(window,whatHappened,panel);
      }
      else
      {
         /* First check if last keypress was less than kMaxTickDelay
            ticks ago - if so, append the current character onto
            our current search string and continue the search. */
         if (((ev->when - gLastTime) < kMaxTickDelay))
         {
            /* Append character only if it doesn't overflow our buffer. */
            if (sKeyDowns.dataLength < kRStringMaxBytes)
            {
               /* Append key onto our previously saved string. */
               sKeyDowns.body[sKeyDowns.dataLength] = theChar;
               ++sKeyDowns.dataLength;
            }
         }
         else
         {
            /* Last keypress was more than kMaxTickDelay ticks ago,
               so treat it as a "new" key to search on. */
            sKeyDowns.body[0] = theChar;
            sKeyDowns.dataLength = 1;
         }

         /* Save time of last keypress for the comparison above. */
         gLastTime = ev->when;
         /* Go select the appropriate item based on our search string. */
         SDPSelectString(panel, &sKeyDowns);
         MyHandleUserSelection(window, kSDPChangedSelection, panel);
      }
   }
}
```

```
void MyHandleUserSelection(WindowPtr window, SDPPanelState whatHappened,
                                SDPPanelHandle panel)
{
   OSErr err;
   SDPSelectionState itsState;

   err = SDPGetPanelSelectionState(panel, &itsState);
   if (err == noErr)
   {
      switch (itsState)
      {
         case kSDPContainerSelected:
            if (whatHappened == kSDPChangedSelection)
            {
               MyDisplayDataForSelection(panel, window);
            }
            break;
         case kSDPRecordSelected :
            if (whatHappened == kSDPChangedSelection)
            {
               MyDisplayDataForSelection(panel, window);
            }
            break;
         case kSDPRecordAliasSelected:
            break;
         case kSDPLockedContainerSelected:
            break;
         case kSDPContainerAliasSelected:
            break;
         case kSDPNothingSelected:
            break;

      }
   }
}

void MyDisplayDataForSelection(SDPPanelHandle panel, WindowPtr window)
{
   OSErr err;
   PackedDSSpec selection;
   DSSpec dss;
   RLI theRLI;
   RecordID rid;
   RString sTempRStr;
```

```
   err = SDPGetPanelSelection(panel, &selection);
   if (err == noErr)
   {
      OCEUnpackDSSpec(&selection, &dss, &rid);
      OCEUnpackRLI(rid.rli,&theRLI);

      /* record name */
      OCECopyRString(rid.local.recordName, &sTempRStr, kRStringMaxBytes);
      /* display the record name now in sTempRStr */
      ...

      /* catalog name */
      OCECopyRString((RStringPtr)theRLI.directoryName, &sTempRStr,
                        kRStringMaxBytes);
      /* display the catalog name now in sTempRStr */
      ...

      /* record type */
      OCECopyRString(rid.local.recordType, &sTempRStr, kRStringMaxBytes);
      /* display the record type now in sTempRStr */
      ...

   }
}

void MyHandleUpdates(WindowPtr window)
{
   GrafPtr savePort;
   SDPPanelHandle panel;

   GetPort(&savePort);
   SetPort(window);
   BeginUpdate(window);
   EraseRect(&window->portRect);

   if (MyGetPanelForThisWindow(window, &panel) == true)
      SDPUpdatePanel(panel, savePort->visRgn);

   EndUpdate(window);
   SetPort(savePort);
}
```

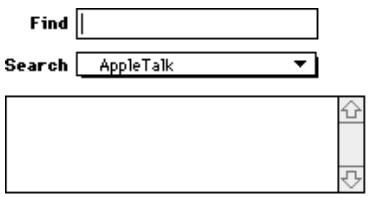
```
void MyHandleActivates(EventRecord *ev)
{
   if ((ev->modifiers & activeFlag) != 0) {
      MyDoActivate((WindowPtr)ev->message);
   }
   else {
      MyDoDeactivate((WindowPtr)ev->message);
   }
}

void MyDoActivate(WindowPtr window)
{
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
   {
      SDPEnablePanel(panel, true);
      SDPUpdatePanel(panel, nil);
   }
}

void MyDoDeactivate(WindowPtr window)
{
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
      SDPEnablePanel(panel, false);
}

void MyHandleSREvt(long message)
{
   extern Boolean gInBackground;
   unsigned long whatMessage;

   whatMessage = message >> 24;

   if (whatMessage == suspendResumeMessage) {
      if ((message & 1) != 0) {
         gInBackground = false;
         SetCursor(&qd.arrow);
         if (FrontWindow()) {
            HiliteWindow(FrontWindow(),true);
            MyDoActivate(FrontWindow());
         }
```

```
      }
      else if (FrontWindow()) {
         gInBackground = true;
         HiliteWindow(FrontWindow(),false);
         MyDoDeActivate(FrontWindow());
      }
   }
}

void MyHandleIdle(EventRecord *ev, WindowPtr window)
{
   OSErr err;
   SDPPanelState whatHappened;
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
      err = SDPPanelEvent(panel, ev, &whatHappened);
}
```

Creating and Disposing of a Find Panel

The Find panel provides an editable text box, a scrolling list, and a pop-up menu that you can place in any window. The text box lets the user enter a search string. The menu specifies the catalogs and volumes to be searched. The list shows AOCE records that have the record types you specify and that have names that start with the text string the user types in the text box. The Find panel allows users to search AOCE catalogs, personal catalogs, and HFS volumes and to select a record. Figure 4-5 shows a Find panel.

Figure 4-5 The Find panel



The preferred user interface to display a Find panel is in a dialog box that also displays a Catalog-Browsing panel. You should provide clearly labeled buttons enabling the user to switch between the find and Catalog-Browsing panels within the dialog box. The AppleMail application included with the AOCE software provides an example of this user interface, which is invoked when the user clicks the Recipients button on a mailer.

The function shown in Listing 4-5 illustrates how to create a Find panel. If the call to the `SDPNewFindPanel` function succeeds, and if there is a `'STR#'` resource available containing help balloon strings for the Find panel, the function installs Balloon Help online assistance. Finally, the function installs an application-defined callback routine for the Find panel to call whenever it is busy, such as while it is searching.

Listing 4-5 Creating a Find panel

```
OSErr MyCreateFindPanel(WindowPtr            window,
                        const RStringPtr     *typesList,
                        unsigned long        typeCount,
                        AuthIdentity         identity,
                        short                helpResourceID)
{
   OSErr status;
   Point panelLoc;
   SDPFindPanelHandle thePanel;

   SetPt(&panelLoc, 0, 0);
   status = SDPNewFindPanel(
         &thePanel,                /* address of panel handle*/
         window,                   /* in this window */
         panelLoc,                 /* at upper left in window */
         kStandardFindLayout,      /* normal layout */
         false, false,             /* initially invisible */
         typesList,                /* types to display */
         typeCount,                /* number of types in list */
         kExactMatch,              /* display only list types */
         identity,                 /* identity of the caller */
         10,                       /* simultaneous searches */
         kSDPFindPanelTextHasFocus, /* text input field focus */
         (long) window             /* reference constant */
         );

   if (status == noErr && helpResourceID != 0)
      status = SDPSetFindPanelBalloonHelp(
                                 thePanel, helpResourceID);

   if (status == noErr)
      status = SDPInstallFindPanelBusyProc(
                                 thePanel, MyArrowSpinnerFunc);
```

```
   return status;
}
```

The code in Listing 4-6 illustrates an application-defined function that disposes of a Find panel. The only feature of this function in addition to calling `SDPDisposeFindPanel` is a check to see if the panel has already been deallocated.

Listing 4-6 Disposing of a Find panel

```
OSErr MyFindPanelDispose(SDPFindPanelHandle thePanel)
{
   OSErr status;

   if (thePanel == NULL)
      status = noErr;
   else {
      status = SDPDisposeFindPanel(thePanel);
      thePanel = NULL;
   }
   return status;
}
```

Standard Catalog Package Reference

This section describes the data types and routines provided by the Standard Catalog Package.

Data Types

The Standard Catalog Package routines use the data types described in this section.

Catalog-Browsing Panel Structure

The Catalog-Browsing panel provides a way for users to locate catalogs, look through them, and select individual records.

The `SDPPanelRecord` structure is the main data type representing the panel. You can examine the contents of any of the fields, but you should not write to any field except the `refCon` field, which you can use for whatever purpose you wish. You might want to look at the contents of the `bounds` field, which stores the rectangle around the panel in the local coordinates of the window that contains the panel. (You provide the `GrafPort`

structure for this window when you call the SDPNewPanel function.) The SDPNewPanel and SDPGetNewPanel functions return handles to SDPPanelRecord structures.

IMPORTANT

The public fields of the SDPPanelRecord structure are followed by private fields. You must not resize the handle to this structure or allocate one yourself. Always use the SDPGetNewPanel function (page 4-34) or SDPNewPanel function (page 4-30) to allocate an SDPPanelRecord structure. ▲

```
struct SDPPanelRecord {
   Rect           bounds;     /* rectangle around panel */
   Boolean        visible;    /* Is panel visible? */
   Boolean        enabled;    /* Is panel enabled? */
   Boolean        focused;    /* Is focus box around panel? */
   Byte           padByte;    /* reserved */
   AuthIdentity   identity;   /* auth ID of caller of panel */
   long           refCon;     /* for your use */
   Rect           listRect;   /* rectangle on scrolling list */
   Rect           popupRect;  /* rectangle around pop-up menu */
   short          numberOfRows;/* rows in scrolling list */
   short          rowHeight;  /* height of list rows (points) */
   Boolean        pabMode;    /* Is panel in personal catalog? */
};
```

Field descriptions

bounds	The rectangle around the panel in the local coordinates of the window that contains the panel. The bounds rectangle includes both the scrolling field and the pop-up menu above it.
visible	A Boolean value indicating controlling whether the panel is currently visible. You can use the SDPShowPanel and SDPHidePanel functions to set this value.
enabled	A Boolean value that specifies whether the panel is currently enabled. You can use the SDPEnablePanel function to set this value.
focused	A Boolean value that specifies whether the panel has a focus rectangle drawn around it to indicate that it is the target of keystroke events.
identity	The authentication identity of the caller of the panel, corresponding to the name and password of the user.
refCon	A reference constant. You can use this field for whatever you wish.
listRect	The rectangle around only the scrolling list.
popupRect	The rectangle around only the pop-up menu.
numberOfRows	The number of lines in the scrolling list.
rowHeight	The height, in printer's points, of each line in the scrolling list.

`pabMode` A Boolean value indicating whether the panel is browsing the user's default personal catalog, rather than a standard catalog.

Find Panel Structure

The Find panel provides a way for users to locate catalogs, look through them, and select individual records.

The `SDPFindPanelRecord` structure is the main data type representing the Find panel. Its fields are reserved for internal use only, except for the `refCon` field, which you can use for whatever purpose you wish. The `SDPNewFindPanel` function returns a handle to an `SDPFindPanelRecord` structure.

IMPORTANT

The public fields of the `SDPFindPanelRecord` structure are followed by private fields. You must not resize the handle to this structure or allocate one yourself. Always use the `SDPNewFindPanel` function (page 4-61) to allocate an `SDPFindPanelRecord` structure. ▲

```
struct SDPFindPanelRecord {
   Point             upperLeft;                /* reserved */
   Boolean           visible;                  /* reserved */
   Boolean           enabled;                  /* reserved */
   Boolean           nowFinding;               /* reserved */
   Byte              padByte;                  /* reserved */
   SDPFindPanelFocus currentFocus;             /* reserved */
   AuthIdentity      identity;                 /* reserved */
   short             simultaneousSearchCount;  /* reserved */
   long              refCon;                   /* for your use */
};
```

Field descriptions

`upperLeft` The upper-left corner of the Find panel in the window's local coordinates.

`visible` A Boolean value indicating whether the Find panel is currently visible. You can use the `SDPShowFindPanel` and `SDPHideFindPanel` functions to set this value.

`enabled` A Boolean value that specifies whether the Find panel is currently enabled. You can use the `SDPEnableFindPanel` function to set this value.

`nowFinding` A Boolean value indicating whether the Find panel is currently busy.

`currentFocus` A constant that specifies whether there is a focus rectangle in the Find panel, and if so, whether it is around the scrolling list in the Find panel or around the text-input field (the Find field; see Figure 4-5 on page 4-18). You can use the `SDSetFindPanelFocus` function to change the location of the focus rectangle.

`identity`	The authentication identity of the caller of the Find panel, corresponding to the name and password of the user.
`simultaneousSearchCount`	The number of catalog searches that can be done simultaneously. See the description of this parameter on page 4-62 for more information.
`refCon`	A reference constant. You can use this field for whatever you wish.

RString List

The `SDPGetCategories` function (page 4-91) and `SDPGetCategoryTypes` function (page 4-92) return handles to lists of items in the form of a packed `RString` list.

```
typedef PackedPathNamePtr *PackedRStringListHandle;
```

A `PackedPathName` structure contains a length plus an array of bytes:

```
#define PackedPathNameHeader\
   unsigned short dataLength;/* excludes the dataLength field */

struct PackedPathName {
   PackedPathNameHeader
   Byte  data[kPathNameMaxBytes - sizeof(unsigned short)];
};
```

The packed `RString` list consists of a `PackedPathName` structure containing packed `RString` structures. Each `RString` structure contains a script code, a length, and a string: Use the AOCE string utility routines described in the chapter "AOCE Utilities" in this book to unpack and manipulate this structure. Use the `OCEDNodeNameCount` function described in that chapter to determine the number of items in a `PackedPathName` structure.

```
#define RStringHeader      \
   CharacterSet charSet;   \
   unsigned short dataLength;

struct RString {
   RStringHeader
   Byte  body[kRStringMaxBytes];    /* place of characters */
};
```

Standard Catalog Package Functions

This section and the following sections describe the routines provided by the AOCE Standard Catalog Package. Many Standard Catalog Package routines require you to

provide an authentication identity as input. The subsection "Authenticating a User," beginning on page 4-25, describes routines that prompt the user for a name and password, authenticate the user, and return the authentication identity number to your application.

The second dialog box allows the user to search for a record by specifying the name or the first part of the name of the record.

The section "Creating, Displaying, and Disposing of a Catalog-Browsing Panel," beginning on page 4-29, provides functions that you can use to display a Catalog-Browsing panel. This panel lets the user browse through HFS catalogs and AOCE catalogs and select a record from a catalog. You can place a Catalog-Browsing panel in any of your application's windows.

The routines in"Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, allow you to process events related to the panel, such as a user selecting an item displayed in the panel.

The section "Creating, Displaying, and Disposing of a Find Panel" on page 4-61 provides functions that you can use to display a Find panel. The Find panel allows the user to search for a record by specifying the name or the first part of the name of the record. The Find panel is similar in implementation to the Catalog-Browsing panel; you can place one in any of your application's windows.

The routines in "Handling Events in a Find Panel," beginning on page 4-75, allow you process events related to the Find panel.

The section "Resolving Aliases," beginning on page 4-85, provides functions that resolve HFS and AOCE aliases of objects in the AOCE catalog system.

The section "Obtaining Icons and Lists of Catalog-Item Categories and Types," beginning on page 4-88, provides functions that return icons for AOCE components such as attributes, records, and catalogs, and functions that return lists of the record types and record categories currently available.

Assembly-Language Interface

To call a Standard Catalog Package routine from assembly language, you must do the following:

1. Push space for the function result and all routine parameters (in Pascal calling-convention order) on the stack.
2. Put in the D0 register a long word consisting of the parameter word count for the routine followed by the routine selector. The parameter word count indicates how many words of parameters you are placing on the stack; for example, if the function has two parameters and each is a pointer, the parameter word count for the function is $0004.
3. Call the Standard Catalog Package trap, $AA5D.

Each routine description in the following sections lists the parameter word count and routine selector for that routine.

Note
The routines described in the section "Obtaining Icons and Lists of Catalog-Item Categories and Types," beginning on page 4-88 use the $AA5C trap and do not require a parameter count. ◆

Authenticating a User

The `SDPPromptForID` function in this section authenticates a user. The function displays dialog boxes that enable the user to enter a name and password, and it returns either a local or specific authentication identity. Authentication identities are described in "Creating an Authentication Identity," beginning on page 4-6. You must provide a valid authentication identity as a parameter to many AOCE functions.

SDPPromptForID

The `SDPPromptForID` function displays a dialog box that allows the user to enter a name and password. The function returns the user's authentication identity.

```
pascal OSErr SDPPromptForID (AuthIdentity *id,
                             ConstStr255Param guestPrompt,
                             ConstStr255Param specificIDPrompt,
                             ConstStr255Param localIDPrompt,
                             const RString *recordType,
                             SDPIdentityKind permittedKinds,
                             SDPIdentityKind *selectedKind,
                             const RecordID *loginFilter,
                             SDPLoginFilterKind filterKind);
```

`id` The authentication identity returned by the function. This identity corresponds to the name and password entered by the user and can be a local identity or a specific identity, depending on the value you specify for the `permittedKinds` parameter and which radio button the user selects. If the user cancels the authentication, the function returns the `userCanceledErr` result code and the ID is undefined. If the user logs on with guest access, the function returns 0 in this parameter.

`guestPrompt`
The prompt string displayed when the user selects the Guest radio button.

`specificIDPrompt`
The prompt string displayed when the user selects the PowerShare account radio button.

`localIDPrompt`
: The prompt string displayed when the user selects the "Fewer Choices" version of the dialog box or when the user selects the Key Chain Access Code radio button.

`recordType`
: The record type of the records that can be authenticated. The function uses this type to determine for which records to accept a name and password in the dialog box. If you specify `nil`, the function uses the user record type. To obtain the value to use for a standard record type, you pass an index value to the utility function `OCEGetIndRecordType`. For example, you can use the index value `kUserRecTypeNum` to get a pointer to the user record type. Other index values are listed in the chapter "AOCE Utilities" in this book.

`permittedKinds`
: A value that specifies whether the user should be allowed to log on with guest access (an authentication ID of 0), as an alternate user (that is, with the password for a specific identity), or with the password for the local identity. Use any combination of the bit masks `kSDPGuestMask`, `kSDPSpecificIdentityMask`, and `kSDPLocalIdentityMask` for this parameter.

`selectedKind`
: A value that indicates which type of identity the function returned: an authentication ID of 0 (guest access), a specific identity (alternate-user access), or a local identity. This parameter can have any one of the following values: `kSDPGuestMask`, `kSDPSpecificIdentityMask`, or `kSDPLocalIdentityMask`.

`loginFilter`
: A pointer to a record ID that specifies a catalog or record name that is initially displayed in the dialog box. You use the `filterKind` parameter to specify what is in this parameter and whether to restrict the user to this catalog or record. Specify `nil` for this parameter to allow any user to log on to any catalog. The function uses this parameter only when the user selects the PowerShare account radio button.

`filterKind`
: A value that indicates which type of information you have provided in the `loginFilter` parameter. If you specify `kSDPRestrictToDirectory` for the `filterKind` parameter, you must specify a catalog in the `loginFilter` parameter. The dialog box displays this catalog and allows only users with records in that catalog to log on. If the record ID you provide in the `loginFilter` parameter includes a record name, the function places that name in the Name text field of the dialog box but doesn't restrict the user to that record.

  If you specify `kSDPRestrictToRecord` for the `filterKind` parameter, you must provide the record ID of a user record for the `loginFilter` parameter, and only the user specified by that record ID can log on.

If you specify `kSDPSuggestionOnly` for the `filterKind` parameter, the dialog box initially displays the catalog and record name you specify in the `loginFilter` parameter but does not restrict the user to that selection.

If you specify `nil` for the `loginFilter` parameter, the function ignores the `filterKind` parameter.

The function uses the `filterKind` parameter only when the user selects the PowerShare account radio button.

DESCRIPTION

The dialog box displayed by the `SDPPromptForID` function lets the user verify a local identity by entering a password, log on with a specific identity by selecting a catalog and then entering a name and password, or log on to a catalog as a guest. The function returns the authentication identity for the user in the `id` parameter. If the name or password entered by the user is not valid, the dialog box prompts the user to reenter the information. If the user fails three times to enter the correct password, the function returns the `kSDPTooManyLoginAttempts` result code. If the user cancels the dialog box, the function returns the `userCanceledErr` result code.

The owner or principal user of a Macintosh computer enters a name and a password during the first attempt to use an AOCE catalog or mailbox. If the user elects access as the principal user and has never set up a local identity, the function guides the user through the process of setting up a local identity. If the user selects the Key Chain Access Code radio button and has previously entered a password to verify the local identity during the current computer session, the function displays a dialog box with the user's name but without a text-entry field for the password. When the user clicks the OK button, the function returns the local identity.

To obtain a local identity without displaying a dialog box, call the `AuthGetLocalIdentity` function. If that function returns an error, then call the `SDPPromptForID` function to allow the user to unlock the local identity or set one up if none exists.

You can use the `loginFilter` and `filterKind` parameters to restrict the dialog box to a single catalog or record, or to set a specific record as the one displayed initially in the dialog box.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0010	$0388

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel button
`kSDPTooManyLoginAttempts`	–1951	User failed three times to enter correct password

SEE ALSO

You can use the `DirEnumerateDirectoriesGet` and `DirEnumerateDirectoriesParse` functions to obtain discriminators for catalogs that are listed in the PowerTalk Setup catalog. You can use the `DirNetSearchADAPDirectoriesGet` and `DirNetSearchADAPDirectoriesParse` functions to obtain discriminators for all catalogs on the network. These functions are all described in the chapter "Catalog Manager" in this book.

You can use the `AuthGetLocalIdentity` function to obtain a local identity without displaying a dialog box. See the chapter "Authentication Manager" in this book for a description of that function.

Sorting a Personal Catalog

The routine in this section operates on personal catalogs.

SDPRepairPersonalDirectory

The `SDPRepairPersonalDirectory` function sorts the contents of a personal catalog according to the current script system.

```
pascal OSErr SDPRepairPersonalDirectory (FSSpec *pd,
                                         Boolean showProgress);
```

`pd` The file system specification structure for the personal catalog you wish to sort.

`showProgress`
A Boolean value indicating whether the Standard Catalog Package should display a progress bar to show the user the progress of the sorting operation.

DESCRIPTION

If the user moves a personal catalog to a computer whose operating system uses a different script system from the one last used to sort the catalog, the personal catalog must be resorted before the Catalog Manager can open it. If the `DirOpenPersonalDirectory` function returns the error `kOCEVersionErr`, you must

call the `SDPRepairPersonalDirectory` function to resort the personal catalog and then call the `DirOpenPersonalDirectory` function again to open the catalog.

If you specify `true` for the `showProgress` parameter, the Standard Catalog Package first displays a dialog box that tells the user there is a problem with the personal catalog and asks if it should correct the problem. If the user elects to correct the problem, the Standard Catalog Package sorts the catalog, displaying a progress bar as it does so. If you specify `false` for the `showProgress` parameter, the Standard Catalog Package sorts the catalog without any feedback to the user.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$1A2C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `DirOpenPersonalDirectory` function is described in the chapter "Catalog Manager" in this book.

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

You can call either the `SDPNewPanel` function (page 4-30) or the `SDPGetNewPanel` function (page 4-34) to create a new panel. The two functions are identical, except that you pass all the parameters to the `SDPNewPanel` function when you call the function, whereas you specify several of the parameters to the `SDPGetNewPanel` function in a resource.

The `SDPNewPanel` and `SDPGetNewPanel` functions allow you to specify a particular container (volume, folder, AOCE catalog, personal catalog, or node) for the panel to display when it opens. After the panel is open, you can use the `SDPSetPath` function (page 4-38) to change the container the panel is displaying.

The other functions in this section, as their names suggest, allow you to hide, show, enable, and disable a panel, redraw a panel, move a panel within your window, and change the size of a panel. When you are finished using a panel or close the window, call the `SDPDisposePanel` function (page 4-50).

Once you have placed a panel in your window, you use the functions in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that take place within the panel.

SDPNewPanel

The `SDPNewPanel` function creates a new Catalog-Browsing panel with the size and location you specify in the window you specify.

```
pascal OSErr SDPNewPanel (SDPPanelHandle *newPanel,
                          WindowPtr window,
                          const Rect *bounds,
                          Boolean visible,
                          Boolean enabled,
                          const PackedRLI *initialRLI,
                          const RStringPtr *typesList,
                          unsigned long typeCount,
                          AuthIdentity identity,
                          DirEnumChoices enumFlags,
                          DirMatchWith matchTypeHow,
                          long refCon);
```

`newPanel` The handle to the new panel created by the function. The `SDPNewPanel` function allocates this handle.

`window` The window into which the panel is to be inserted.

`bounds` The bounds for the panel, including both the pop-up menu above the scrollable list and the list itself (see Figure 4-4), in the local coordinates of the window you specify in the `window` parameter. You can calculate the minimum width of a panel as

```
5 * FontInfo.widMax + 42
```

and the minimum height as

```
Max(64, 3 * FontHeight) + fontHeight + 18
```

where

```
fontHeight = FontInfo.ascent + FontInfo.descent +
             FontInfo.Leading.
```

If the bounds you specify do not enclose an integral number of items in the scrollable list, the `SDPNewPanel` function shortens the panel slightly to prevent the last item in the list from being cropped.

`visible` A Boolean value that specifies whether the panel is initially visible. Set this parameter to `true` if you want the panel to be visible. You can use the `SDPShowPanel` and `SDPHidePanel` functions to show and hide a panel.

`enabled` A Boolean value that specifies whether the panel is initially enabled. Set this parameter to `true` to enable the panel. You can use the `SDPEnablePanel` function to enable and disable the panel.

`initialRLI`
A pointer to the volume, folder, AOCE catalog, personal catalog, or node that the panel should display initially. If you specify `nil` for this parameter, the panel displays a list of volumes and AOCE catalogs. Specify –1 for this parameter to open a Personal-Catalog panel rather than the standard Catalog-Browsing panel. You can use the `OCEGetDirectoryRootPackedRLI` function to get the location of the root of all catalogs (represented on the desktop by the Catalogs icon).

`typesList` A pointer to an array, each item of which is of type `RStringPtr`, providing a list of the types of records you want the panel to display. The panel displays only records of the types you specify in this list. The `matchTypeHow` parameter specifies how the `SDPNewPanel` function interprets the types list. To display aliases of dNodes, you must include a record type of "DNode" (the value `kDNodeRecTypeNum`) and include the value `kEnumAliasMask` in the `enumFlags` parameter.

`typeCount` The number of record types in your types list.

`identity` The authentication identity of the caller. Specify 0 for guest access.

`enumFlags` A mask value that specifies what sort of information you want the Standard Catalog Package to display in the panel. You can set this field to any combination of the following constants. The constant `kEnumAllMask` combines all of the other values.

`kEnumDistinguishedNameMask`
records

`kEnumAliasMask`
aliases

`kEnumPseudonymMask`
pseudonyms

`kEnumDNodeMaskd`
Nodes

`kEnumInvisibleMask`
invisible entities

`kEnumAllMask`all
entities

`matchTypeHow`
A constant that specifies how the function interprets the types list. The possible values for this parameter are described in the following function description.

`refCon` A reference constant. The function places this value in the `refCon` field of the `SDPPanelRecord` structure pointed to by the `newPanel` parameter. You can use the `refCon` parameter for whatever you wish.

DESCRIPTION

You use the `SDPNewPanel` function to create a new Catalog-Browsing panel in your window. You specify the location and size of the panel, and can specify that the panel is to be initially visible or invisible and initially enabled or disabled. You can use the `enumFlags` parameter to specify which types of entities (records, dNodes, aliases, or pseudonyms) the panel displays.

The `typesList` and `matchTypeHow` parameters let you specify what types of records the panel should display. The possible values for the `matchTypeHow` parameter are as follows:

```
enum {
   kMatchAll,
   kExactMatch,
   kBeginsWith,
   kEndingWith,
   kContaining
};
typedef unsigned char DirMatchWith;
```

Constant descriptions

`kMatchAll`	Display all record types. The function ignores the `typesList` and `typeCount` parameters.
`kExactMatch`	Display only the record types in the types list.
`kBeginsWith`	Display only records whose types begin with the string pointed to by the `typesList` parameter. The `typesList` parameter can point to only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kEndingWith`	Display only records whose types end with the string pointed to by the `typesList` parameter. The `typesList` parameter can point to only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kContaining`	Display only records whose types contain the string pointed to by the `typesList` parameter. The `typesList` parameter can point to only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.

You can use the `SDPGetCategories` and `SDPGetCategoryTypes` functions to determine what record types are available and use that information to let the user select which types of records the Catalog-Browsing panel should display. Then you can use the `typesList` parameter to restrict the panel to those record types.

Subsequently, you can use the other routines in this and the following section to hide and show the panel, enable and disable it, and handle events that occur within the panel. If you wish to specify some parameters in a resource rather than when you call the function, use the `SDPGetNewPanel` function instead.

Listing 4-3 on page 4-9 illustrates the use of the `SDPNewPanel` function to create a new panel.

SPECIAL CONSIDERATIONS

The window's font, text size, and text style affect how the panel appears when it is created.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0015	$0064

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPGetNewPanel` function, described next, does the same thing as the `SDPNewPanel` function but allows you to specify several of the parameters in a resource.

You can use the `SDPHidePanel` function (page 4-43) and `SDPShowPanel` function (page 4-44) to hide and show a Find panel.

You can use the `SDPEnablePanel` function (page 4-45) to enable and disable the Find panel.

Use the other routines in this section to dispose of, draw, move, resize, and otherwise manipulate panels. Use the routines in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that occur in the panel and to determine what record the user selected.

The `PackedRecordLocationInfo` structure is described in the chapter "Catalog Manager" in this book.

You can use the `OCEGetDirectoryRootPackedRLI` function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

You can use the `SDPPromptForID` function (page 4-25) to get a local or specific identity.

You can use the `SDPGetCategories` function (page 4-91) to obtain a list of all the record-type categories currently available and the `SDPGetCategoryTypes` function (page 4-92) to list all the types of records included in a specific category.

SDPGetNewPanel

The `SDPGetNewPanel` function creates a new panel in the window you specify with the size and location specified in a resource of type `'panl'`.

```
pascal OSErr SDPGetNewPanel (SDPPanelHandle *newPanel,
                             short resourceID,
                             WindowPtr window,
                             const PackedRLI *initialRLI,
                             AuthIdentity identity);
```

`newPanel` A handle to the new panel created by the function.

`resourceID`
The resource ID of the panel resource, which contains several parameters for use by the function.

`window` A pointer to the window into which the panel is to be inserted.

`initialRLI`
A pointer to the volume, folder, AOCE catalog, node, or personal catalog that the dialog box should display initially. If you specify `nil` for this parameter, the dialog box displays a list of volumes and AOCE catalogs. Specify –1 for this parameter to open a Personal-Catalog panel rather than the standard Catalog-Browsing panel.

`identity` The authentication identity of the caller. Specify 0 for guest access.

DESCRIPTION

The `SDPGetNewPanel` function creates a new panel and returns its handle, given a resource with the following Rez type:

```
type 'panl' {
   rect;                            /* bounds parameter */
   byte     invisible, visible;     /* visible parameter */
   byte     disabled, enabled;      /* enabled parameter */
   longint;                         /* enumFlags parameter */
   integer;                         /* matchTypeHow parameter */
   longint;                         /* refCon parameter */
   integer = $$CountOf(TypeIdArray);/* typeCount parameter */
   array TypeIdArray {
    integer;                        /* typesList parameter
                                       ('rtyp' resource ID
                                         for each type) */
   };
};

#define kSDPPanelResourceType    'panl'
```

See the `SDPNewPanel` function on page 4-30 for descriptions of these parameters.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0009	$0065

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPNewPanel` function (page 4-30) does the same thing as the `SDPGetNewPanel` function, but you specify all of the parameters when you call the `SDPNewPanel` function rather than using a panel resource.

Use the other routines in this section to dispose of, hide, show, enable, disable, draw, move, and resize panels. Use the routines in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that occur in the panel and to determine what record the user selected.

The `PackedRecordLocationInfo` structure is described in the chapter "Catalog Manager" in this book.

You can use the `OCEGetDirectoryRootPackedRLI` function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

You can use the `SDPPromptForID` function (page 4-25) to get a specific or local identity.

SDPInstallPanelBusyProc

The `SDPInstallPanelBusyProc` function installs an application-defined routine that the panel calls when it is busy.

```
pascal OSErr SDPInstallPanelBusyProc (SDPPanelHandle panel,
                                      PanelBusyProc busyProc);
```

`panel` A handle for the panel for which you want to install a panel-busy callback routine.

`busyProc` A pointer to your callback routine. To remove a callback routine that you installed previously, specify `nil` for this parameter.

DESCRIPTION

If you use the `SDPInstallPanelBusyProc` function to install a panel-busy callback routine, the Catalog-Browsing panel calls your routine whenever the panel is busy. For example, if the user double-clicks a dNode in the panel, the panel calls your callback routine while it searches the dNode.

The panel passes to your callback routine the handle to the panel and a Boolean value that indicates whether the panel is busy. You can use your callback routine to provide feedback to the user that indicates that the panel is busy. One good way to do this is to display the Standard Catalog Package's spinning arrow icon.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0079

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The panel-busy callback routine is described on page 4-94.

SDPSetPanelBalloonHelp

The `SDPSetPanelBalloonHelp` function enables Balloon Help for the panel.

```
pascal OSErr SDPSetPanelBalloonHelp (SDPPanelHandle panel,
                                     short balloonHelpID);
```

`panel` A handle for the panel for which you want to enable Balloon Help.

`balloonHelpID`
The resource ID of a `'STR#'` resource that contains Balloon Help strings for the panel.

DESCRIPTION

You must use the `SDPSetPanelBalloonHelp` function to provide text strings for Balloon Help and to activate Balloon Help for the panel. You must provide two text strings, one for each element of the panel, in the following order:

1. the scrolling list
2. the pop-up menu

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0078

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Balloon Help is described in the "Help Manager" chapter of *Inside Macintosh: More Macintosh Toolbox*.

SDPSetIdentity

The `SDPSetIdentity` function changes the authentication identity used by the panel.

```
pascal OSErr SDPSetIdentity (SDPPanelHandle panel,
                             AuthIdentity identity);
```

`panel` A handle for the panel for which you want to change the caller's authentication identity.

`identity` The new authentication identity. Specify 0 for guest access.

DESCRIPTION

You can use the `SDPSetIdentity` function to change the authentication identity of the user while the user is browsing catalogs. If the new identity does not have read privileges for the dNode being browsed, the panel moves up levels in the catalog until it

reaches a level at which the user has read privileges and the function returns the `kOCEReadAccessDenied` result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0073

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter
`kOCEReadAccessDenied`	–1540	User does not have read privileges.

SEE ALSO

The `SDPPromptForID` function (page 4-25) prompts the user for a password and returns an authentication identity.

SDPSetPath

The `SDPSetPath` function causes the panel to display the contents of the container specified by the packed record location information structure (`packedRLI` data type) you specify.

```
pascal OSErr SDPSetPath (SDPPanelHandle panel,
                         const PackedRLI *prli);
```

`panel` The panel on which you want this function to act.

`prli` A pointer to the record location information for the container whose contents you want the panel to display. Specify `nil` for this parameter to display a list of volumes and AOCE catalogs. You can use the `OCEGetDirectoryRootPackedRLI` function to get the location of the root of all catalogs (represented on the desktop by the Catalogs icon).

DESCRIPTION

When you call the `SDPSetPath` function, the panel's pop-up menu displays the container (volume, folder, AOCE catalog, personal catalog, or node) you specify in the `prli` parameter, and the scrollable list displays the contents of that container. Whereas

the SDPNewPanel and SDPGetNewPanel functions let you specify which container the panel displays when it opens, the SDPSetPath function lets you change the container displayed by a panel that is already open. If the container you specify in the prli parameter is the one currently being browsed in the panel, then this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0070

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

The PackedRecordLocationInfo structure is described in the chapter "Catalog Manager" in this book.

You can use the OCEGetDirectoryRootPackedRLI function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

To select an item in the scrollable list of the panel, use the SDPSelectString function (page 4-42).

Use the SDPNewPanel (page 4-30) or SDPGetNewPanel (page 4-34) functions to specify which container a panel displays initially.

SDPGetPathLength

The SDPGetPathLength function returns the size of the buffer required to hold the pathname of the current item in the panel's pop-up menu.

```
pascal OSErr SDPGetPathLength (SDPPanelHandle panel,
                               unsigned short *pathNameLength);
```

panel The panel for which you want information.

`pathNameLength`
A pointer to the length, in bytes, of the `PackedRLI` structure representing the current pathname. You must allocate this pointer.

DESCRIPTION

The `SDPGetPathLength` function returns the number of bytes required to hold the `PackedRLI` structure that represents the current pathname of the container (volume, folder, AOCE catalog, dNode, or personal catalog) in the pop-up menu of the panel. You can allocate a buffer of this size and pass a pointer to it in the `prli` parameter of the `SDPGetPath` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0075

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the value returned by this function to determine how big a buffer to allocate for the `SDPGetPath` function, described next.

SDPGetPath

The `SDPGetPath` function returns the pathname of the current item in the panel's pop-up menu.

```
pascal OSErr SDPGetPath (SDPPanelHandle panel,
                         PackedRLI *prli,
                         short *dsRefNum);
```

`panel` The panel for which you want information.

`prli` A pointer to a buffer you provide to hold the current pathname. Specify `nil` for this parameter if you want the function to return only the `dsRefNum` parameter.

`dsRefNum` A pointer to a location to hold a reference number if the item is a personal catalog. You must allocate this pointer. Specify `nil` for this parameter if you want the function to return only the `prli` parameter.

DESCRIPTION

The `SDPGetPath` function returns the current pathname of the container (volume, folder, AOCE catalog, dNode, or personal catalog) in the pop-up menu of the panel and—if the container is a personal catalog—a reference number. You can call the `SDPGetPathLength` function to determine the size of buffer required for the pathname. The `SDPGetPath` function returns the pathname in packed RLI format.

If you want the function to return either the pathname or the reference number, but not both, specify `nil` for the parameter you do not want returned.

You can use the record location information (`prli`) returned by this function to call Catalog Manager functions, such as `DirGetDirectoryInfo` or `DirFindValue`, that return information about the container. You can use the value returned in the `dsRefNum` parameter if you immediately want to call a Catalog Manager function that takes a personal-catalog reference number as input. Once you have called the `SDPPanelEvent` function or the `SDPDisposePanel` function, you can no longer use this reference number. In that case, you must use the `DirOpenPersonalDirectory` function to open the personal catalog and obtain a new reference number.

SPECIAL CONSIDERATIONS

Because the personal-catalog reference number returned in the `dsRefNum` parameter is valid only while the personal catalog is open, this value may not be valid after you call the `SDPPanelEvent` function and is never valid after you call the `SDPDisposePanel` function.

The pathname of a container returned by the `SDPGetPath` function is not necessarily the same as the pathname you get by using the `SDPGetPanelSelection` function to obtain location information for an item the user has selected in that container. This discrepancy results from the fact that the item might be an alias or information card. The `SDPGetPanelSelection` function returns the location of the original item, not that of the alias or information card itself.

The `SDPGetPath` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0076

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the value returned by the `SDPGetPathLength` function (page 4-39) to determine the size of the buffer to allocate for the pathname.

The `packedRLI` structure and Catalog Manager routines are described in the chapter "Catalog Manager" in this book.

You call the `SDPPanelEvent` function (page 4-52) to handle events that take place in the panel. You call the `SDPDisposePanel` function (page 4-50) to dispose of a panel that you no longer need.

You can use the `SDPGetPanelSelection` function (page 4-58) to obtain location information for a panel item that a user has selected. You can use the `SDPGetPanelSelectionState` function (page 4-55) to determine the nature of the selected item.

SDPSelectString

The `SDPSelectString` function scrolls to and highlights the item in the panel that best matches the string you specify.

```
pascal OSErr SDPSelectString (SDPPanelHandle panel,
                              const RString *name);
```

`panel`	The panel on which you want this function to act.
`name`	A pointer to the string you want the function to match.

DESCRIPTION

When you call the `SDPSelectString` function, the panel's scrollable list displays and highlights the item that best matches the string you specify in the `name` parameter. The matching algorithm is the same one used by the Standard File dialog box when the user types a string: the function matches characters starting at the beginning of the string. If the function cannot match all of the characters in the string, it selects the item in the list that follows the string alphabetically. For example, if the panel contains the items Andy, Atticus, Bernice, Bruce, and Freda, and you specify the string "Bru", the function highlights "Bruce". If you specify the string "Charlie", the function highlights "Freda". The `SDPSelectString` function acts on the list at the current level; that is, it does not change the path represented by the item selected in the pop-up menu.

You could use this function, for example, to display the item the user had selected the last time the panel was opened. If the string you specify in the `name` parameter matches the item currently being selected, then this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0074

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

To set a specific path in the pop-up menu of the panel, use the `SDPSetPath` function (page 4-38).

SDPHidePanel

The `SDPHidePanel` function hides a panel.

```
pascal OSErr SDPHidePanel (SDPPanelHandle panel);
```

`panel` The panel you wish to hide.

DESCRIPTION

If a panel is visible, this function makes it invisible by hiding the menu, erasing and invalidating the list's rectangle, and causing the list to not be drawn. If the panel is already hidden, this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0067

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowPanel` function (described next) to display a panel that you have hidden.

SDPShowPanel

The `SDPShowPanel` function displays a panel.

```
pascal OSErr SDPShowPanel (SDPPanelHandle panel);
```

`panel` The panel you wish to display.

DESCRIPTION

You can hide a panel by setting the `visible` parameter in the `SDPNewPanel` or in the `'panl'` resource of the `SDPGetNewPanel` function to `false` or by calling the `SDPHidePanel` function. If the panel is hidden, the `SDPShowPanel` function makes it visible. If the panel is already visible, this function does nothing. Use the `SDPUpdatePanel` function to handle an update event.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0068

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You can specify that a panel is to be initially hidden when you create it with the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34).

Use the `SDPHidePanel` function (page 4-43) to hide a panel.

Use the `SDPUpdatePanel` function (page 4-47) to handle an update event.

SDPEnablePanel

The `SDPEnablePanel` function enables or disables a panel.

```
pascal OSErr SDPEnablePanel (SDPPanelHandle panel,
                             Boolean enable);
```

`panel`	The panel you wish to enable or disable.
`enable`	A Boolean value that specifies whether you wish to enable or disable the panel. Specify `true` to enable the panel.

DESCRIPTION

This function disables the list and menu so that they do not accept any commands, or enables a disabled panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0069

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you call the `SDPNewPanel` function (page 4-30) to create a panel, you specify whether it should be enabled or disabled initially.

SDPSetFocus

The `SDPSetFocus` function draws or removes a focus rectangle around a panel.

```
pascal OSErr SDPSetFocus (SDPPanelHandle panel,
                          Boolean focus);
```

`panel` The panel for which you wish to draw or remove a focus rectangle.

`focus` A Boolean value that specifies whether you wish to draw or remove the rectangle. Specify `true` to draw the rectangle and `false` to remove it.

DESCRIPTION

When the user clicks in a panel or uses the Tab key to select it (assuming you support this feature), you can use the `SDPSetFocus` function to draw a heavy border around the panel. This border, called a focus rectangle (or focus box), indicates to the user that the panel is active and that any keypress commands will be applied to the panel.

If you never call the `SDPSetFocus` function, the Standard Catalog Package highlights the user's current selection regardless of whether the user is working in the panel or in your window. However, once you call the `SDPSetFocus` function, the Standard Catalog Package removes highlighting in the panel whenever you remove the focus rectangle from the panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0077

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SDPUpdatePanel

The SDPUpdatePanel function draws all or part of a panel.

```
pascal OSErr SDPUpdatePanel (SDPPanelHandle panel,
                             RgnHandle theRgn);
```

panel	The panel you wish to draw.
theRgn	The region of the window you wish to draw. Any component of the panel that intersects this region is redrawn. For example, if any portion of the list of the panel intersects this region, the Standard Catalog Package redraws the whole list. Pass nil in this parameter to draw the entire panel.

DESCRIPTION

You must call the SDPUpdatePanel function to draw a panel in response to an update event. (Do not pass an update event to the SDPPanelEvent function.) You can also use the SDPUpdatePanel function to draw a panel when your application needs to redraw its window because of some activity other than an update event.

In response to an update event, you should first call the BeginUpdate routine, which takes a window pointer as a parameter. Use the value in the visRgn field of the window's grafPort structure for the theRgn parameter to the SDPUpdatePanel function.

Note that the SDPUpdatePanel function does not use the region specified by theRgn as a clipping region. To clip the drawing region, you must first call the SetClip routine. (Remember to save the old clipping region so that you can restore it when you have finished clipping.)

If you have hidden the panel (either by setting the visible parameter in the SDPNewPanel function or SDPGetNewPanel function to false or by calling the SDPHidePanel function), then the SDPUpdatePanel function does not display the panel. To display a hidden panel, you must call the SDPShowPanel function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006A

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowPanel` function (page 4-44) to display a panel that you have hidden with the `SDPNewPanel` function (page 4-30), `SDPGetNewPanel` function (page 4-34), or `SDPHidePanel` function (page 4-43).

Use the `BeginUpdate` routine to begin the process of updating your window. The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how to respond to update events. The `BeginUpdate` routine is described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Use the `SetClip` routine to set a clipping region. The `SetClip` routine is described in *Inside Macintosh: Imaging With QuickDraw.*

SDPMovePanel

The `SDPMovePanel` function moves the panel to a location you specify.

```
pascal OSErr SDPMovePanel (SDPPanelHandle panel,
                           short h,
                           short v);
```

`panel`	The panel you wish to move.
`h`	The horizontal coordinate to which you want to move the upper-left corner of the panel, in the local coordinates of the panel's window.
`v`	The vertical coordinate to which you want to move the upper-left corner of the panel, in the local coordinates of the panel's window.

DESCRIPTION

Use the `SDPMovePanel` function to move the upper-left corner of the panel to (h, v), given in local coordinates of the panel's window. You set the original location of the panel with the `bounds` parameter in the `SDPNewPanel` function or `SDPGetNewPanel` function.

If you have hidden the panel (either by setting the `visible` parameter in the `SDPNewPanel` function or `SDPGetNewPanel` function to `false` or by calling the `SDPHidePanel` function), then the `SDPMovePanel` function does not display the panel. To display a hidden panel, you must call the `SDPShowPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34) to set the original location of the panel.

Use the `SDPShowPanel` function (page 4-44) to display a panel that you have hidden with the `SDPNewPanel` function, `SDPGetNewPanel` function, or `SDPHidePanel` function (page 4-43).

SDPSizePanel

The `SDPSizePanel` function resizes the panel to a size you specify.

```
pascal OSErr SDPSizePanel (SDPPanelHandle panel,
                           short width,
                           short height);
```

`panel` The panel you wish to resize.

`width` The width, in QuickDraw coordinates, that you want to use for the panel. You must specify a value for this parameter—it has no default value. You can calculate the minimum width of a panel as

```
5 * FontInfo.widMax + 42
```

`height` The height, in points, that you want to use for the panel. You must specify a value for this parameter—it has no default value. You can calculate the minimum height of a panel as

```
Max(64, 3 * FontHeight) + fontHeight + 18
```

where

```
fontHeight = FontInfo.ascent + FontInfo.descent +
             FontInfo.Leading.
```

DESCRIPTION

The `SDPSizePanel` function resizes the panel to have the specified width and height (keeping the upper-left corner in a fixed position). You set the original size of the panel with the `bounds` parameter in the `SDPNewPanel` function or `SDPGetNewPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34) to set the original size of the panel.

You can use the `GetFontInfo` routine described in *Inside Macintosh: Imaging With QuickDraw,* to determine the font size.

SDPDisposePanel

The `SDPDisposePanel` function deallocates all of the data structures associated with a panel.

```
pascal OSErr SDPDisposePanel (SDPPanelHandle panel);
```

`panel` The handle of the panel you wish to deallocate.

DESCRIPTION

Call this function when you are finished using a panel.

SPECIAL CONSIDERATIONS

A personal-catalog reference number returned by the `SDPGetPath` function is no longer valid after you call the `SDPDisposePanel` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0066

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPGetPath` function (page 4-40) returns a personal-catalog reference number that is no longer valid after you dispose of the panel.

Call the `SDPNewPanel` function (page 4-30) or the `SDPGetNewPanel` function (page 4-34) to open a new panel.

Handling Events in a Catalog-Browsing Panel

The routines in this section let you find out what action the user has taken in the Catalog-Browsing panel (or Personal-Catalog panel) and handle the resulting event.

If an event takes place in the Catalog-Browsing panel, you call the first function in this section, `SDPPanelEvent`, to handle the event. You must also pass regular null events to the `SDPPanelEvent` function. This function returns a value that tells you if the user has double-clicked a record or has otherwise changed the selection in the panel. You can use the `SDPGetPanelSelectionSize` (page 4-57) and `SDPGetPanelSelection` (page 4-58) functions to determine what item the user has selected. If the user has double-clicked a record, you can use these functions to determine what record the user has double-clicked.

You must call the `SDPUpdatePanel` function (page 4-47) to handle update events.

If the user double-clicks any item other than a record, the panel opens the item and displays its contents. If you want to provide a method other than double-clicking for a user to open an item, such as an Open item in a menu or a button in your window, you can use the `SDPOpenSelectedItem` function (page 4-59) to do so.

If you are implementing menu items or controls in addition to those in the panel, you can use the `SDPGetPanelSelectionState` function (page 4-55) to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record once to highlight it, you can enable a button labeled Choose. If the user has highlighted a container, you can change the button to read Open.

SDPPanelEvent

The `SDPPanelEvent` function handles events intended for the panel.

```
pascal OSErr SDPPanelEvent (SDPPanelHandle panel,
                            const EventRecord *theEvent,
                            SDPPanelState *whatHappened);
```

`panel` The panel in which the event occurred.

`theEvent` The event record for the event.

`whatHappened`
A return value that tells you how the `SDPPanelEvent` function handled the event.

DESCRIPTION

Because the panel is a modeless dialog box, the Event Manager passes all events that occur in the panel to you. If you determine that the event occurred in the panel, you can use the `SDPPanelEvent` function to handle the event.

You should use this function to handle mouse-up and mouse-down events in the panel, any key-down and auto-key events you want the panel to process, disk-insert events for which the panel is the intended target, and activate events for the panel's window. It treats suspend and resume events like activate events and all other events as null events. You must call the `SDPUpdatePanel` function to handle update events.

Before you call the `SDPPanelEvent` function for a key-down event, you should provide the appropriate feedback to the user. For example, if you provide a button labeled Open for opening an item in the panel and the user presses the Return key, you should highlight the Open button before calling the `SDPPanelEvent` function.

You must regularly provide the panel with null events so that it can update the scrollable list when the user moves up or down in the catalog hierarchy and so it can enumerate catalogs when appropriate to do so.

The `whatHappened` parameter can have any of the following values:

```
enum {
   kSDPProcessed,
   kSDPSelectedAnItem,
   kSDPChangedSelection
};

typedef unsigned short SDPPanelState;
```

Constant descriptions

`kSDPProcessed` The event resulted in no state change.

`kSDPSelectedAnItem`
The user wants to select the currently highlighted record. The function returns this value, for example, when a user double-clicks a record. When you are displaying the Personal-Catalog panel, the `SDPPanelEvent` function also returns this value when the user double-clicks an alias of a container.

`kSDPChangedSelection`
This is a good time to check the state of the panel. The function returns this value when item selected in the panel has changed because the user clicked a new item (which may mean that no item is selected), clicked in a menu, pressed a Command-key combination, or inserted a disk. The function also returns this value from the first null event you send to a panel after creating the panel.

If you are displaying the Personal-Catalog panel and the user attempts to open an alias of a container, the panel does not open the container, because the contents of that container are not within the user's default personal catalog. In that case, the `SDPPanelEvent` function returns the value `kSDPSelectedAnItem` in the `whatHappened` parameter. You can distinguish this case from that of the user selecting a record by calling the `SDPGetPanelSelectionState` function, which returns the value `kSDPRecordSelected` or `kSDPRecordAliasSelected` if the user selected a record or the alias of a record, but returns the value `kSDPContainerAliasSelected` if the user selected the alias of a container.

To display the contents of a container when the user tries to open the alias of a container in the Personal-Catalog panel, you can call the `SDPGetPanelSelectionSize` and `SDPGetPanelSelection` functions to get the location of the container, then close the Personal-Catalog panel and open the Catalog-Browsing panel. Use the location of the container as the value of the `initialRLI` parameter when you call the `SDPNewPanel` function or the `SDPGetNewPanel` function to open the new panel. Alternatively, you can get faster performance at the expense of more memory use by keeping both a Personal-Catalog panel and a Catalog-Browsing panel open and hiding the one not in use. Then when the user attempts to open the alias of a container in the Personal-Catalog panel, you can hide the Personal-Catalog panel, display the Catalog-Browsing panel, and use the `SDPSetPath` function to display the contents of the appropriate container.

SPECIAL CONSIDERATIONS

You cannot assume that a personal-catalog reference number returned by the `SDPGetPath` function is valid after you call the `SDPPanelEvent` function.

You must call the `SDPUpdatePanel` function to handle update events.

Be sure to pass mouse-down and mouse-up events to the `SDPPanelEvent` function. If you don't do so, the panel cannot respond to double clicks.

The `SDPPanelEvent` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0071

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Call the `SDPUpdatePanel` function (page 4-47) to handle update events.

Use the `SDPGetPanelSelectionState` function (described next) to find out what the user has selected in the panel.

Call the `SDPGetPanelSelectionSize` function (page 4-57) to determine the size of the packed `DSSpec` structure of a selected item in the panel and the `SDPGetPanelSelection` function (page 4-58) to get a pointer to the packed `DSSpec` structure.

Call the `SDPNewPanel` function (page 4-30) or the `SDPGetNewPanel` function (page 4-34) to open a new panel. Call the `SDPHidePanel` (page 4-43) and `SDPShowPanel` (page 4-44) functions to hide and display open panels.

Use the `SDPSetPath` function (page 4-38) to display the contents of a specific container in an open panel.

The `SDPGetPath` function (page 4-40) returns a personal-catalog reference number that you can no longer assume to be valid after you call the `SDPPanelEvent` function.

SDPGetPanelSelectionState

The `SDPGetPanelSelectionState` function tells you what the user has selected in the panel.

```
pascal OSErr SDPGetPanelSelectionState (SDPPanelHandle panel,
                                        SDPSelectionState *itsState);
```

`panel` The panel on which you want this function to act.

`itsState` The state of the panel.

DESCRIPTION

The parameter `itsState` can return any of the following values:

```
enum {
   kSDPNothingSelected,
   kSDPLockedContainerSelected,
   kSDPContainerSelected,
   kSDPRecordSelected,
   kSDPRecordAliasSelected,
   kSDPContainerAliasSelected
};

typedef unsigned short SDPSelectionState;
```

Constant descriptions

`kSDPNothingSelected`
Nothing is currently selected.

`kSDPLockedContainerSelected`
The user has selected a container (a volume, folder, AOCE catalog, dNode, or personal catalog), but doesn't have access privileges.

`kSDPContainerSelected`
The user has selected a volume, folder, AOCE catalog, dNode, or personal catalog.

`kSDPRecordSelected`
The user has selected a record.

`kSDPRecordAliasSelected`
The user has selected an alias of a record.

`kSDPContainerAliasSelected`
The user has selected an alias of a container.

If the user has selected a record or the alias of a record, you can use the `SDPGetPanelSelectionSize` function to determine the size of the `PackedDSSpec` structure and the `SDPGetPanelSelection` function to get the `PackedDSSpec` structure for that record. If the `SDPGetPanelSelectionState` function returns the value `kSDPContainerSelected` or `kSDPLockedContainerSelected`, you can use these functions to determine which container the user has selected.

If you are displaying the Catalog-Browsing panel and the user selects an alias of a container, the `SDPGetPanelSelectionState` function returns the value `kSDPContainerAliasSelected`, but otherwise the panel behaves as if the user had selected the container itself. However, if you are displaying the Personal-Catalog panel and the user attempts to open an alias of a container, the panel does not open the container. In this case, you can call the `SDPGetPanelSelectionSize` and `SDPGetPanelSelection` functions to get the location of the container, then close (or hide) the Personal-Catalog panel and open (or show, if it's already open) the Catalog-Browsing panel to display the contents of that container.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006E

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

See the description of the `SDPPanelEvent` function (page 4-52) for more information on handling user selections in panels.

Call the `SDPGetPanelSelectionSize` function (page 4-57) to determine the size of the packed `DSSpec` structure of a selected record and the `SDPGetPanelSelection` function (page 4-58) to get a pointer to the packed `DSSpec` structure.

Use the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34) to open a new panel.

SDPGetPanelSelectionSize

The `SDPGetPanelSelectionSize` function returns the size of the `PackedDSSpec` structure containing the packed record ID of the currently selected record or container.

```
pascal OSErr SDPGetPanelSelectionSize (SDPPanelHandle panel,
                                       unsigned short *dsSpecSize);
```

`panel` The panel on which you want this function to act.

`dsSpecSize`

A pointer to the size of the `PackedDSSpec` structure containing the record ID and location information for the record or container (dNode, AOCE catalog, personal catalog, volume, or folder) that the user has selected.

DESCRIPTION

If the `SDPGetPanelSelectionState` function returns the value `kRecordSelected` or `kSDPRecordAliasSelected`, you can use the `SDPGetPanelSelectionSize` and `SDPGetPanelSelection` functions to determine which record the user selected. If the `SDPGetPanelSelectionState` function returns the value `kContainerSelected`, `kLockedContainerSelected`, or `kContainerAliasSelected`, these functions tell you which container the user has selected.

You can use the length returned in the `dsSpecSize` parameter to allocate a buffer for a `PackedDSSpec` structure. You can then use a pointer to this buffer for the `selection` parameter when you call the `SDPGetPanelSelection` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0072

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetPanelSelectionState` function (page 4-55) to determine what (a record, container, locked container, or nothing) the user has selected in the panel.

Use the `SDPGetPanelSelection` function, described next, to get the `DSSpec` structure identifying the record or container the user has selected.

SDPGetPanelSelection

The `SDPGetPanelSelection` function returns a `DSSpec` structure for the record or container the user has selected in the panel.

```
pascal OSErr SDPGetPanelSelection (SDPPanelHandle panel,
                                   PackedDSSpec *selection);
```

`panel` The panel on which you want this function to act.

`selection` A pointer to a `PackedDSSpec` structure, which contains location information for a volume, folder, dNode, personal catalog, or record. You must allocate memory and provide this pointer before you call the function.

DESCRIPTION

If the `SDPGetPanelSelectionState` function returns the value `kRecordSelected`, you can use the `SDPGetPanelSelection` function to determine which record the user selected. If the `SDPGetPanelSelectionState` function returns the value `kContainerSelected` or `kLockedContainerSelected`, the user has selected a volume, folder, AOCE catalog, dNode, or personal catalog, and the `SDPGetPanelSelection` function tells you the location of that container.

If the `SDPGetPanelSelectionState` function returns the value `kRecordAliasSelected`, the `SDPGetPanelSelection` function returns the `DSSpec` structure for the record referred to by the alias, not for the alias itself. Similarly, if the `SDPGetPanelSelectionState` function returns the value `kContainerAliasSelected`, the `SDPGetPanelSelection` function tells you the location of the container referred to by the alias.

Before you call the `SDPGetPanelSelection` function, you can call the `SDPGetPanelSelectionSize` function to get the size of the `PackedDSSpec` structure and use it to allocate the buffer for the `selection` parameter.

SPECIAL CONSIDERATIONS

The pathname of a container returned by the `SDPGetPath` function is not necessarily the same as the pathname you get by using the `SDPGetPanelSelection` function to obtain location information for an item the user has selected in that container. This discrepancy results from the fact that the item might be an alias or information card. The

SDPGetPanelSelection function returns the location of the original item, not that of the alias or information card itself.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006F

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Use the SDPGetPanelSelectionState function (page 4-55) to determine whether the user has selected a record, a container, a locked container, or an alias.

Use the SDPGetPanelSelectionSize function (page 4-57) to determine the size of the packed DSSpec structure.

The SDPGetPath function (page 4-40) returns the current pathname of the container in the pop-up menu of the panel.

SDPOpenSelectedItem

The SDPOpenSelectedItem function simulates a double-click on an item that the user has selected in the panel.

```
pascal OSErr SDPOpenSelectedItem (SDPPanelHandle panel,
                                  SDPPanelState *whatHappened);
```

panel The panel on which you want this function to act.

whatHappened
The result of the double click. This parameter can return the value kSDPProcessed, kSDPSelectedAnItem, or kSDPChangedSelection.

DESCRIPTION

If the user has selected a container (that is, a volume, folder, AOCE catalog, personal catalog, or node), then the SDPOpenSelectedItem function opens that container and returns the value kSDPChangedSelection in the whatHappened parameter. If the user has selected a record or an alias of a record, then the function returns the value

kSDPSelectedAnItem in this parameter. If the user has selected an alias of a container in the Catalog-Browsing panel, the SDPOpenSelectedItem function returns the value kSDPChangedSelection and opens that container just as if the user had selected the container itself. However, if the user has selected an alias of a container in the Personal-Catalog panel, the function returns the value kSDPSelectedAnItem and does not open the container. If the user has selected a locked container or there is no item selected, the function returns kSDPProcessed.

The Catalog-Browsing panel allows the user to double-click an item but provides no other user interface for opening a container or choosing a record. You can use the SDPOpenSelectedItem function to provide a way for the user to open a container or choose a record in the panel without double-clicking. For example, you can provide a button adjacent to the panel and call the SDPOpenSelectedItem function when the user clicks the button. To implement such a feature, you should call the SDPGetPanelSelectionState function each time either the SDPOpenSelectedItem function or the SDPPanelEvent function returns the value kSDPChangedSelection. If the user has selected a container or an alias of a container, you can enable the Open button. If there is no item selected or the user has selected a locked container, you can disable the button. If the user has selected a record, you can change the button to read "Choose" or whatever is appropriate for your application.

If the SDPOpenSelectedItem function returns the value kSDPSelectedAnItem for the Catalog-Browsing panel, you can assume the item is a record and you can use the SDPGetPanelSelectionSize function to determine the size of the packed DSSpec structure and the SDPGetPanelSelection function to get the DSSpec for that record. For the Personal-Catalog panel, you must first determine whether the item is a record or an alias of a container. If it's a record, you can get the DSSpec structure just as in the case of the Catalog-Browsing panel. If the item is an alias of a container, you can switch to a Catalog-Browsing panel to display the contents of the container.

If the SDPOpenSelectedItem function returns the value kSDPChangedSelection, you should call the SDPGetPanelSelectionState function to determine how to label your buttons and whether to enable them. If the function returns kSDPProcessed, you need take no further action.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006D

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPPanelEvent function (page 4-52) for mouse-up and mouse-down events in the panel to find out if the user changed the selection.

Call the SDPGetPanelSelectionSize function (page 4-57) to determine the size of the packed DSSpec structure of a selected item and the SDPGetPanelSelection function (page 4-58) to get a handle to the packed DSSpec structure.

Creating, Displaying, and Disposing of a Find Panel

The routines in this section create and display a Find panel. You can call the SDPNewFindPanel function (described next) to create a new Find panel.

The other functions in this section, as their names suggest, allow you to hide, show, enable, and disable a Find panel, redraw a Find panel, and move a Find panel within your window. When you are finished using a Find panel or close the window, call the SDPDisposeFindPanel function (page 4-75).

Once you have placed a Find panel in your window, you use the functions in "Handling Events in a Find Panel," beginning on page 4-75, to handle events that take place within the Find panel.

SDPNewFindPanel

The SDPNewFindPanel function creates a new Find panel in the location you specify in the window you specify.

```
pascal OSErr SDPNewFindPanel (SDPFindPanelHandle *newPanel,
                  WindowPtr window,
                  Point upperLeft,
                  short layoutResourceID,
                  Boolean visible,
                  Boolean enabled,
                  const RStringPtr *typesList,
                  long typeCount,
                  DirMatchWith matchTypeHow,
                  AuthIdentity identity,
                  short simultaneousSearchCount,
                  SDPFindPanelFocus initialFocus,
                  long refCon);
```

`newPanel` The address of the handle to the new Find panel created by the function. The `SDPNewFindPanel` function allocates this handle.

`window` A pointer to the window into which the Find panel is to be inserted.

`upperLeft` The upper-left corner, in the window's local coordinates, of the Find panel.

`layoutResourceID`
The resource ID of a Find-panel layout resource (type `kSDPFindPanelResourceType`) that specifies the layout of the Find panel. You can specify `kStandardFindLayout` as the resource ID to use a standard layout for the Find panel.

`visible` A Boolean value that specifies whether the Find panel is to be initially visible. Set this parameter to `true` if you want the Find panel to be visible. You can use the `SDPShowFindPanel` and `SDPHideFindPanel` functions to hide and show a Find panel.

`enabled` A Boolean value that specifies whether the Find panel is to be initially enabled. Set this parameter to `true` to enable the Find panel. You can use the `SDPEnableFindPanel` function to enable and disable the Find panel.

`typesList` A pointer to an array, each item of which is of type `RStringPtr`, providing a list of the types of records you want the Find panel to display. The Find panel displays only records of the types you specify in this list. The `matchTypeHow` parameter specifies how the `SDPNewFindPanel` function interprets the types list.

`typeCount` The number of record types in your types list.

`matchTypeHow`
`A constant that specifies how the function interprets the types list. The possible values for this parameter are described in the following function description.`

`identity` The authentication identity of the caller. Specify 0 for guest access.

`simultaneousSearchCount`
The number of catalog searches that may be done simultaneously. If you specify a larger number for this parameter, the searches are generally faster but require more memory. Each search requires 4600 bytes. The function does not use more memory than is necessary; for example, if you specify 10 for this parameter but there are only two catalogs to search, the function uses only the memory necessary to search those two catalogs. Note that the Find panel can perform simultaneous searches of catalogs only; it cannot search more than one volume at a time.

`initialFocus`
A constant that specifies whether there should be a focus rectangle in the Find panel, and if so, whether it should be around the scrolling list in the Find panel or around the text-input field (the Find field; see Figure 4-5 on page 4-18). You can use the `SDSetFindPanelFocus` function to change the location of the focus rectangle.

`refCon` A reference constant. The function places this value in the `refCon` field of the `SDPPanelRecord` structure pointed to by the `newPanel` parameter. You can use the `refCon` parameter for whatever you wish.

DESCRIPTION

You use the `SDPNewFindPanel` function to create a new Find panel in your window. You specify the location of the Find panel and can specify that it is to be initially visible or invisible and initially enabled or disabled.

The `typesList` and `matchTypeHow` parameters let you specify what types of records the Find panel should display. The possible values for the `matchTypeHow` parameter are as follows:

```
enum {
   kMatchAll,
   kExactMatch,
   kBeginsWith,
   kEndingWith,
   kContaining
};
```

Constant descriptions

`kMatchAll`	Display all record types. The function ignores the `typesList` and `typeCount` parameters.
`kExactMatch`	Display only the record types in the types list.
`kBeginsWith`	Display only records whose types begin with the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kEndingWith`	Display only records whose types end with the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kContaining`	Display only records whose types contain the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.

You can use the `SDPGetCategories` and `SDPGetCategoryTypes` functions to determine what record types are available and use that information to let the user select which types of records the Find panel should display. Then you can use the `typesList` parameter to restrict the panel to those record types.

In the `layoutResourceID` parameter you provide the resource ID of a Find-panel layout resource, defined by the following Rez type:

```
type 'find' {
   pstring;       /* "Find" text */
   align word;
   pstring;       /* "Search" text */
   align word;
```

```
   array [5] {    /* specifications for text-entry box label,
                     search menu label, text-entry box,
                     search menu, scrolling list */
      integer sysFont, appFont, portFont;
      integer;    /* face */
      integer;    /* size */
      rect;       /* bounds */
   };
};
#define kSDPFindPanelResourceType   'find'
```

Field descriptions

`"Find" text` The label for the text-entry field. You must supply a value for this field; there is no default label.

`"Search" text` The label for the search menu. You must supply a value for this field; there is no default label.

`array` An array of specifications for the elements of the Find panel in the following order: The label for the text-entry box, the label for the search menu, the text-entry box, the search menu, and the scrolling list. The specifications include the following information:

- `font` A choice of system font, application font, or graphics port font.
- `face` The QuickDraw font style.
- `size` The type size, in points.
- `bounds` The boundary, in local coordinates, of the element.

Subsequently, you can use the other routines in this and the following section to hide and show the Find panel, enable and disable it, and handle events that occur within the Find panel.

You may specify any of the following values for the `initialFocus` parameter:

```
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};

typedef unsigned short SDPFindPanelFocus;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0014	$08FC

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You can use the `SDPHideFindPanel` function (page 4-67) and `SDPShowFindPanel` function (page 4-68) to hide and show a Find panel.

You can use the `SDPEnableFindPanel` function (page 4-69) to enable and disable the Find panel.

You can use the `SDSetFindPanelFocus` function (page 4-69) to change the location of the focus rectangle.

Use the other routines in this section to dispose, draw, and move Find panels. Use the routines in "Handling Events in a Find Panel," beginning on page 4-75, to handle events that occur in the Find panel and to determine what record the user selected.

You can use the `SDPPromptForID` function (page 4-25) to get a specific or local identity.

You can use the `SDPGetCategories` function (page 4-91) to obtain a list of all the record-type categories currently available and the `SDPGetCategoryTypes` function (page 4-92) to list all the types of records included in a specific category.

SDPInstallFindPanelBusyProc

The `SDPInstallFindPanelBusyProc` function installs an application-defined routine that the Find panel calls when it is busy.

```
pascal OSErr SDPInstallFindPanelBusyProc (
                                    SDPFindPanelHandle findPanel,
                                    FindPanelBusyProc busyProc);
```

`findPanel` A handle for the Find panel for which you want to install a Find-panel-busy callback routine.

`busyProc` A pointer to your callback routine. To remove a callback routine that you installed previously, specify `nil` for this parameter.

DESCRIPTION

If you use the `SDPInstallFindPanelBusyProc` function to install a Find-panel-busy callback routine, the Find panel calls your routine whenever the panel is busy. For

example, if the user initiates a search, the Find panel calls your callback routine while it conducts the search.

The Find panel passes to your callback routine the handle to the panel and a Boolean value that indicates whether the panel is busy. You can use your callback routine to provide feedback to the user that indicates that the Find panel is busy. One good way to do this is to display the Standard Catalog Package's spinning arrow icon.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$090C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The Find-panel-busy callback routine is described on page 4-95.

SDPSetFindPanelBalloonHelp

The `SDPSetFindPanelBalloonHelp` function enables Balloon Help for the Find panel.

```
pascal OSErr SDPSetFindPanelBalloonHelp (
                                    SDPFindPanelHandle findPanel,
                                    short balloonHelpID);
```

`findPanel` A handle for the panel for which you want to enable Balloon Help.

`balloonHelpID`
The resource ID of a `'STR#'` resource that contains Balloon Help strings for the Find panel.

DESCRIPTION

You must use the `SDPSetFindPanelBalloonHelp` function to provide text strings for Balloon Help and to activate Balloon Help for the Find panel. You must provide three text strings, one for each element of the Find panel, in the following order:

1. the text-input field (the Find field)
2. the search menu
3. the scrolling list

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$090A

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Balloon Help is described in the chapter "Help Manager" of *Inside Macintosh: More Macintosh Toolbox*.

SDPHideFindPanel

The `SDPHideFindPanel` function hides a Find panel.

```
pascal OSErr SDPHideFindPanel (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel you wish to hide.

DESCRIPTION

If a Find panel is visible, this function makes it invisible by hiding the menu, erasing and invalidating the list's rectangles for the list and text fields, and causing the panel to not be drawn. If the Find panel is already hidden, this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0903

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowFindPanel` function (described next) to display a Find panel that you have hidden.

SDPShowFindPanel

The `SDPShowFindPanel` function displays a Find panel.

```
pascal OSErr SDPShowFindPanel (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel you wish to display.

DESCRIPTION

You can hide a Find panel by setting the `visible` parameter in the `SDPNewFindPanel` to `false` or by calling the `SDPHideFindPanel` function. If the Find panel is hidden, the `SDPShowFindPanel` function makes it visible. If the Find panel is already visible, this function does nothing. Use the `SDPUpdateFindPanel` function to handle an update event.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0902

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you create a Find panel with the `SDPNewFindPanel` function (page 4-61), you can specify that it be initially hidden.

Use the `SDPHideFindPanel` function (page 4-67) to hide a Find panel.

Use the `SDPUpdateFindPanel` function (page 4-72) to handle an update event.

SDPEnableFindPanel

The `SDPEnableFindPanel` function enables or disables a Find panel.

```
pascal OSErr SDPEnableFindPanel (SDPFindPanelHandle findPanel,
                                 Boolean enabled);
```

`findPanel`	The Find panel you wish to enable or disable.
`enabled`	A Boolean value that specifies whether you wish to enable or disable the Find panel. Specify `true` to enable the Find panel.

DESCRIPTION

This function disables the list, menu, and text field so that they do not accept any commands or text, or enables a disabled Find panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0905

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you call the `SDPNewFindPanel` function (page 4-61) to create a Find panel, you specify whether it should be enabled or disabled initially.

SDPSetFindPanelFocus

The `SDPSetFindPanelFocus` function changes the focus rectangle for a Find panel.

```
pascal OSErr SDPSetFindPanelFocus (SDPFindPanelHandle findPanel,
                                   SDPFindPanelFocus newFocus);
```

`findPanel` The Find panel for which you wish to change the focus rectangle.

`newFocus` A constant that specifies whether there should be a focus in the Find panel, and if so, whether it should be the scrolling list in the Find panel or the text-input field (the Find field; see Figure 4-5 on page 4-18).

DESCRIPTION

When the user uses the Tab key to select the Find panel (assuming you support this feature), you can use the `SDPSetFindPanelFocus` function to specify which portion of the Find panel (the scrolling list or the text-entry field) is the focus; that is, to which portion of the Find panel any keyboard equivalents are applied. If the scrolling list is the focus, the Find panel draws a heavy border (called a *focus rectangle*) around the list. If the text-entry field is the focus, the Find panel displays a blinking insertion point in the field.

If you want to integrate the action of the focus in the Find panel with one in your application window, you can use the `SDPSetFindPanelFocus` function to control whether the focus is in the Find panel and which element has the focus.

For example, if you have three text-entry fields in your application window and the user presses the Tab key repeatedly, you can use this function to cycle the focus rectangle sequentially through your three fields, then to the text-entry field in the Find panel, then the scrolling list, and then back to the first of your text fields.

To implement this feature, you must interpret events before calling the `SDPFindPanelEvent` function. Otherwise, when the user is working in the Find panel and you call the `SDPFindPanelEvent` function in response to a press of the Tab key, the Find panel toggles the focus between the text-entry field and the scrolling list.

You may specify any of the following values for the `newFocus` parameter:

```
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};
typedef unsigned short SDPFindPanelFocus;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0906

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you call the `SDPNewFindPanel` function (page 4-61) to create a Find panel, you specify the initial focus rectangle for the Find panel.

You should call the `SDPFindPanelEvent` function (page 4-76) to handle events that occur in your window when the Find panel is open.

SDPSetFindIdentity

The `SDPSetFindIdentity` function changes the authentication identity used by the Find panel.

```
pascal OSErr SDPSetFindIdentity (SDPFindPanelHandle findPanel,
                                 AuthIdentity identity)
```

`findPanel` The Find panel for which you want to change the caller's authentication identity.

`identity` The new authentication identity. Specify 0 for guest access.

DESCRIPTION

You can use the `SDPSetFindIdentity` function to change the authentication identity of the user without closing the Find panel and opening a new one.

SPECIAL CONSIDERATIONS

If a search is in progress when you call the `SDPSetFindIdentity` function, the Find panel continues to use the old identity until the search is complete.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$090B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPPromptForID` function (page 4-25) prompts the user for a password and returns an authentication identity.

SDPUpdateFindPanel

The `SDPUpdateFindPanel` function draws all or part of a Find panel.

```
pascal OSErr SDPUpdateFindPanel (SDPFindPanelHandle findPanel,
                                 RgnHandle theRgn);
```

`findPanel` The Find panel you wish to draw.

`theRgn` The region of the window you wish to draw. Any component of the Find panel that intersects this region is redrawn. For example, if any portion of the list of the Find panel intersects this region, the Standard Catalog Package redraws the whole list. Pass `nil` in this parameter to draw the entire Find panel.

DESCRIPTION

You must call the `SDPUpdateFindPanel` function to draw a Find panel in response to an update event. (Do not pass an update event to the `SDPFindPanelEvent` function.)

You can also use the `SDPUpdateFindPanel` function to draw a Find panel when your application needs to redraw its window because of some activity other than an update event.

In response to an update event, you should first call the `BeginUpdate` routine, which takes a window pointer as a parameter. Then use the value in the `visRgn` field of the window's `grafPort` structure for the `theRgn` parameter of the `SDPUpdateFindPanel` function.

Note that the `SDPUpdateFindPanel` function does not use the region specified by the parameter `theRgn` as a clipping region. To clip the drawing region, you must first call the `SetClip` routine. (Remember to save the old clipping region so that you can restore it when you have finished clipping.)

If you have hidden the Find panel (either by setting the `visible` parameter in the `SDPNewFindPanel` function to `false` or by calling the `SDPHideFindPanel` function), then the `SDPUpdateFindPanel` function does not display the Find panel. To display a hidden Find panel, you must call the `SDPShowFindPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0901

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowFindPanel` function (page 4-68) to display a Find panel that you have hidden with the `SDPNewFindPanel` function (page 4-61) or `SDPHideFindPanel` function (page 4-67).

Use the `BeginUpdate` routine to begin the process of updating your window. The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how to respond to update events. The `BeginUpdate` routine is described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Use the `SetClip` routine to set a clipping region. The `SetClip` routine is described in *Inside Macintosh: Imaging With QuickDraw.*

SDPMoveFindPanel

The `SDPMoveFindPanel` function moves the Find panel to a location you specify.

```
pascal OSErr SDPMoveFindPanel (SDPFindPanelHandle findPanel,
                               short h,
                               short v);
```

`findPanel`	The Find panel you wish to move.
`h`	The horizontal coordinate to which you want to move the upper-left corner of the Find panel, in the local coordinates of the Find panel's window.
`v`	The vertical coordinate to which you want to move the upper-left corner of the Find panel, in the local coordinates of the Find panel's window.

DESCRIPTION

Use the `SDPMoveFindPanel` function to move the upper-left corner of the Find panel to (h, v), given in local coordinates of the Find panel's window. You set the original location of the Find panel with the `upperLeft` parameter in the `SDPNewFindPanel` function.

If you have hidden the Find panel (either by setting the `visible` parameter in the `SDPNewFindPanel` function to `false` or by calling the `SDPHideFindPanel` function), then the `SDPMoveFindPanel` function does not display the Find panel. To display a hidden Find panel, you must call the `SDPShowFindPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0904

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewFindPanel` function (page 4-61) to set the original location of the Find panel.

Use the SDPShowFindPanel function (page 4-68) to display a Find panel that you have hidden with the SDPNewFindPanel function (page 4-61) or SDPHideFindPanel function (page 4-67).

SDPDisposeFindPanel

The SDPDisposeFindPanel function deallocates all of the data structures associated with a Find panel.

```
pascal OSErr SDPDisposeFindPanel (SDPFindPanelHandle findPanel);
```

findPanel The handle of the Find panel you wish to deallocate.

DESCRIPTION

Call this function when you have completely finished using a Find panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FD

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPNewFindPanel function (page 4-61) to open a new Find panel.

Handling Events in a Find Panel

The routines in this section let you find out what action the user has taken in the Find panel and handle the resulting event. When you receive an event for a window in your application that contains a Find panel, you must first determine whether it took place in the panel. For mouse-down events, you can check the coordinates of the event to see if it was in the Find panel. You must keep track of where the user is working to know how to handle key-down events; for example, you can place a focus rectangle in the Find panel

or around the content portion of the window, according to the last location of a mouse-down event. Then you can assume that any key-down event pertains to the portion of the window inside the focus rectangle.

If an event takes place in the Find panel, you call the first function in this section, `SDPFindPanelEvent`, to handle the event. You must also pass regular null events to the `SDPFindPanelEvent` function. This function returns a value that tells you how the function handled the event. If the user has double-clicked a record, for example, the function returns the value `kSDPSelectedAFindItem` and you can call the `SDPGetFindPanelSelectionSize` (page 4-80) and `SDPGetFindPanelSelection` (page 4-82) functions to determine what record the user has selected.

You must call the `SDPUpdateFindPanel` function (page 4-72) to handle update events.

You can use the `SDPStartFind` function (page 4-83) to implement a Find button. If you do so, you should also provide a Cancel button, which you can implement by calling the `SDPStopFind` function (page 4-84).

If you are implementing menu items or controls in addition to those in the Find panel, you can use the `SDPGetFindPanelState` function (page 4-79) to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record once to highlight it, you can enable a button labeled Choose.

SDPFindPanelEvent

The `SDPFindPanelEvent` function handles events intended for the Find panel.

```
pascal OSErr SDPFindPanelEvent (SDPFindPanelHandle findPanel,
                                const EventRecord *event,
                                SDPFindPanelResult *whatHappened);
```

`findPanel` The Find panel in which the event occurred.

`event` The event record for the event.

`whatHappened`
A return value that tells you how the `SDPFindPanelEvent` function handled the event.

DESCRIPTION

Because the Find panel is in your window, the Event Manager passes all events that occur in the Find panel to you. If you determine that the event occurred in the Find panel, you can use the `SDPFindPanelEvent` function to handle the event.

You should use this function to handle mouse-up and mouse-down events in the Find panel, any key-down and auto-key events you want the Find panel to process, and activate events for the Find panel's window. It treats all other events as null events. You must call the `SDPUpdateFindPanel` function to handle update events.

Before you call the `SDPFindPanelEvent` function for a key-down event, you should provide the appropriate feedback to the user. For example, if you provide a button labeled Open for opening an item in the Find panel and the user presses the Return key, you should highlight the Open button before calling the `SDPFindPanelEvent` function.

You must regularly provide the Find panel with null events so that it can search for the records the user specifies.

The `whatHappened` parameter can have any of the following values:

```
enum {
   kSDPSelectedAFindItem,
   kSDPFindSelectionChanged,
   kSDPFindCompleted,
   kSDPTextStateChanged,
   kSDPFocusChanged,
   kSDPSelectionAndFocusChanged,
   kSDPMenuChanged,
   kSDPSelectionAndMenuChanged,
   kSDPProcessedFind
};
typedef unsigned short SDPFindPanelResult;
```

Constant descriptions

`kSDPSelectedAFindItem`
: The user wants to choose a record. The function returns this value, for example, when a user double-clicks a record or clicks a record once and presses Return or Enter.

`kSDPFindSelectionChanged`
: The item selected in the Find panel has changed because the user clicked a new item (which may mean that no item is selected), pressed an arrow key, or typed the beginning letters of the name of the record (“type-selecting” the record).

`kSDPFindCompleted`
: The Find panel has completed a search. When the Find text-entry field is the focus and the user presses the Return or Enter key, or when you call the `SDPStartFind` function, the Find panel starts a search. Each time you pass an event (including null events) to the `SDPFindPanelEvent` function, the Find panel continues the search for a short time and then returns control to your application. The first time you call the `SDPFindPanelEvent` function after the search is complete, the function returns the value `kSDPFindCompleted`.

`kSDPTextStateChanged`
: The user has entered text in a previously empty Find field or has deleted all text from the field.

`kSDPFocusChanged`
: The focus has moved from the Find text field to the scrolling list or vice versa.

`kSDPSelectionAndFocusChanged`
: The user has changed the focus rectangle to the scrolling list by clicking a new item.

`kSDPMenuChanged`
: The user has changed the item chosen in the Search menu. The Search menu lists the catalogs and volumes available for searching.

`kSDPSelectionAndMenuChanged`
: The user has changed both the item chosen in the search menu and the item selected in the scrolling list.

`kSDPProcessedFind`
: The event resulted in no state change.

SPECIAL CONSIDERATIONS

The `SDPFindPanelEvent` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0900

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Call the `SDPUpdateFindPanel` function (page 4-72) to handle update events.

Call the `SDPGetFindPanelSelectionSize` function (page 4-80) to determine the size of the packed `DSSpec` structure of a selected item and the `SDPGetFindPanelSelection` function (page 4-82) to get a pointer to the packed `DSSpec` structure.

You can call the `SDPStartFind` function (page 4-83) to initiate a search and the `SDPStopFind` function (page 4-84) to cancel one.

SDPGetFindPanelState

The `SDPGetFindPanelState` function tells you what action the user has taken in the Find panel.

```
pascal OSErr SDPGetFindPanelState (SDPFindPanelHandle findPanel,
                                   SDPFindPanelState *itsState);
```

`findPanel` The Find panel on which you want this function to act.

`itsState` The state of the Find panel.

DESCRIPTION

The Find panel (see Figure 4-5 on page 4-18) does not provide buttons or menu items to let the user choose records or initiate searches; you must provide these buttons and menu items. You can use the `SDPGetFindPanelState` function to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record to highlight it, you can enable a button labeled Choose. If the user has entered text in the Find field, you can enable a button labeled Find.

You can use the following bit masks to test the value returned in the `itsState` parameter:

```
enum {
   kSDPItemIsSelectedBit,
   kSDPFindTextExistsBit
};

/* values of SDPFindPanelState */
enum {
   kSDPItemIsSelectedMask = 1<<kSDPItemIsSelectedBit,
   kSDPFindTextExistsMask = 1<<kSDPFindTextExistsBit
};
typedef unsigned short SDPFindPanelState;
```

Constant descriptions

`kSDPItemIsSelectedMask`
The user has selected a record or the alias of a record.

`kSDPFindTextExistsMask`
The user has entered text in the Find field.

If the user has selected a record or the alias of a record, you can use the `SDPGetFindPanelSelectionSize` function to determine the size of the `PackedDSSpec` structure for the record and the `SDPGetFindPanelSelection` function to get the `PackedDSSpec` structure.

If the user has entered text in the Find field, you can activate a button labeled Find. If the user clicks your Find button, you should call the `SDPStartFind` function to start the search. You should also provide a Stop button, which you can implement by calling the `SDPStopFind` function. Also, if the user presses the Return or Enter key during a search, the Find panel stops the search. Therefore, you should highlight the Stop button during a search to indicate that it is the default button.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0907

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

See the description of the `SDPFindPanelEvent` function (page 4-76) for more information on handling user selections in panels.

Call the `SDPGetFindPanelSelectionSize` function (described next) to determine the size of the `PackedDSSpec` structure of a selected record and the `SDPGetFindPanelSelection` function (page 4-82) to get a pointer to the packed `DSSpec` structure.

SDPGetFindPanelSelectionSize

The `SDPGetFindPanelSelectionSize` function returns the size of the `PackedDSSpec` structure containing the packed record ID of the currently selected record in the Find panel.

```
pascal OSErr SDPGetFindPanelSelectionSize (
                                    SDPFindPanelHandle findPanel,
                                    unsigned short *size);
```

`findPanel` The Find panel on which you want this function to act.

`size` A pointer to the size of the `PackedDSSpec` structure containing the record ID and location information for the record that the user has selected.

DESCRIPTION

If the `SDPGetFindPanelState` function returns the value `kSDPItemIsSelectedMask`, you can use the `SDPGetFindPanelSelectionSize` and `SDPGetFindPanelSelection` functions to determine which record the user selected. If the user has selected an alias of a record, these functions tell you which record the alias is for.

You can use the length returned in the `dsSpecSize` parameter to allocate a buffer to hold a `PackedDSSpec` structure. You can then use a pointer to this buffer as the value of the `selection` parameter when you call the `SDPGetFindPanelSelection` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0908

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetFindPanelState` function (page 4-79) to determine if the user has selected a record in the Find panel.

Use the `SDPGetFindPanelSelection` function, described next, to get the `DSSpec` structure identifying the record the user has selected.

SDPGetFindPanelSelection

The `SDPGetFindPanelSelection` function returns a `DSSpec` structure for the record the user has selected in the Find panel.

```
pascal OSErr SDPGetFindPanelSelection (
                                    SDPFindPanelHandle findPanel,
                                    PackedDSSpec *selection);
```

`findPanel` The Find panel on which you want this function to act.

`selection` A pointer to a `PackedDSSpec` structure, which contains location information for a record. You must allocate memory and provide this pointer before you call the function.

DESCRIPTION

If the `SDPGetFindPanelState` function returns the value `kSDPItemIsSelectedMask`, you can use the `SDPGetFindPanelSelection` function to determine which record the user selected.

If the user has selected the alias of a record, the `SDPGetFindPanelSelection` function returns the `DSSpec` structure for the record referred to by the alias, not for the alias itself.

Before you call the `SDPGetFindPanelSelection` function, you can call the `SDPGetFindPanelSelectionSize` function to get the size of the `PackedDSSpec` structure and use it to allocate the buffer for the `selection` parameter.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0909

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetFindPanelState` function (page 4-79) to determine whether the user has selected a record.

Use the `SDPGetFindPanelSelectionSize` function (page 4-80) to determine the size of the packed `DSSpec` structure.

SDPStartFind

The `SDPStartFind` function searches for the record or records the user specifies in the Find text field of the Find panel.

```
pascal OSErr SDPStartFind (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel on which you want this function to act.

DESCRIPTION

The Find panel allows the user to enter text in the Find field and then press Return or Enter to initiate a search but provides no button to start a search. The *Macintosh Human Interface Guidelines* suggest that you should provide clearly labeled buttons to let the user start and cancel searches. You can use the `SDPStartFind` function to provide a button (or other interface) that lets the user start a search for records.

The `SDPStartFind` function returns control to your application as soon as it initiates the search. Each time you call the `SDPFindPanelEvent` function, the Find panel continues the search for a short time before returning control to you. When it completes the search, the `SDPFindPanelEvent` function returns the value `kSDPFindCompleted` in the `whatHappened` parameter.

You can use the `SDPStopFind` function to cancel a search.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FE

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPFindPanelEvent function (page 4-76) to determine when the SDPStartFind function has completed a search.

Call the SDPStopFind function (described next) to cancel a search.

SDPStopFind

The SDPStopFind function cancels a search that you initiated with the SDPStartFind function.

```
pascal OSErr SDPStopFind (SDPFindPanelHandle findPanel);
```

findPanel The Find panel on which you want this function to act.

DESCRIPTION

When you use the SDPStartFind function to initiate a search for records, you should enable a Stop button. If the user clicks this button, you should call the SDPStopFind function to stop the search. Also, if the user presses the Return or Enter key during a search, the Find panel stops the search. Therefore, you should highlight the Stop button during a search to indicate that it is the default button.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FF

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the `SDPStartFind` function (page 4-83) to initiate a search.

Resolving Aliases

The functions in this section resolve HFS and AOCE catalog-system aliases to catalog objects. The user can use the Finder to create an alias for any item in the catalog system, including records, dNodes, catalogs, personal catalogs, and so forth. If the alias is an HFS file, as in the case of a file-system alias for a personal catalog, you can use the `SDPResolveAliasFile` function (described next) to resolve the alias. If the alias is contained in a record, you can use the `SDPResolveAliasDSSpec` function (page 4-87) to resolve it.

SDPResolveAliasFile

The `SDPResolveAliasFile` function resolves a file-system alias of an AOCE catalog object.

```
pascal OSErr SDPResolveAliasFile (FSSpecPtr fileSpec,
                                  PackedDSSpecHandle resolvedDSSpec,
                                  AuthIdentity identity,
                                  Boolean mayPromptUser);
```

`fileSpec` The file specification structure of the alias file you wish to resolve.

`resolvedDSSpec`
A handle to a packed `DSSpec` structure that contains the resolved alias. You must allocate a handle of any size and provide it to the function. The function resizes the handle as necessary and uses it to return the resolved alias of you.

`identity` The authentication identity of the caller.

`mayPromptUser`
A Boolean value that specifies whether the function should present dialog boxes to the user as necessary. If you set this parameter to `true`, the function displays any dialog boxes that are necessary to complete the resolution of the alias; for example, the user might be requested to insert a disk or log on to a file server. If you set this parameter to `false`, the function does not present the user with any dialog boxes but returns with an error if it can't resolve the alias.

DESCRIPTION

A file containing an AOCE catalog-system alias has the `isAlias` bit set in the file's Finder flags field and has one of the following file types:

File type	Constant
`'pabt'`	`kPersonalCatalogFileType`
`'bust'`	`kBusinessCardFileType`
`'dirt'`	`kCatalogFileType`
`'dnod'`	`kDNodeFileType`
`'rcrd'`	`kRecordFileType`

If it is appropriate to present the user with dialog boxes, specify `true` for the `mayPromptUser` parameter to make sure that the function can resolve the alias. If your function is running in the background or for some other reason it is not appropriate to display dialog boxes, set this parameter to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$0E74

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

HFS aliases and the Finder flags field are described in the chapter "Finder Interface" in the book *Inside Macintosh: Macintosh Toolbox Essentials.*

To resolve an alias in an AOCE catalog record, use the `SDPResolveAliasDSSpec` function, described next.

SDPResolveAliasDSSpec

The `SDPResolveAliasDSSpec` function resolves an alias of an AOCE catalog-system alias that is contained in a `PackedDSSpec` structure.

```
pascal OSErr SDPResolveAliasDSSpec (
                                PackedDSSpecHandle theAliasDSSpec,
                                AuthIdentity identity,
                                Boolean mayPromptUser);
```

`theAliasDSSpec`
A handle to the `PackedDSSpec` structure of the alias you wish to resolve. The function returns, in this same handle, the fully resolved alias of the record to which that alias refers.

`identity` The authentication identity of the caller.

`mayPromptUser`
A Boolean value that specifies whether the function should present dialog boxes to the user as necessary. If you set this parameter to `true`, the function displays any dialog boxes that are necessary to complete the resolution of the alias; for example, the user might be requested to log on to a catalog server. If you set this parameter to `false`, the function does not present the user with any dialog boxes but returns with an error if it can't resolve the alias.

DESCRIPTION

When the enumeration specification structure returned by the `DirEnumerateGet` function indicates that a record contains an alias, that record includes an attribute of type `kAliasAttrTypeBody`. The attribute value of such an attribute is a packed `DSSpec` structure. The `SDPResolveAliasDSSpec` function returns the `PackedDSSpec` structure of the fully resolved record to which that alias refers.

If it is appropriate to present the user with dialog boxes, specify `true` for the `mayPromptUser` parameter to make sure that the function can resolve the alias. If your function is running in the background or for some other reason it is not appropriate to display dialog boxes, set this parameter to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$0E75

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

To resolve an HFS file containing an alias of an AOCE catalog object, use the `SDPResolveAliasFile` function (page 4-85).

Obtaining Icons and Lists of Catalog-Item Categories and Types

When you use the Catalog-Browsing panel or Find panel to let the user select records, you might want to display a list of types or categories of records that the user can search for, a list of records the user has selected, or other information about the catalog structure and contents. The functions in this section return icons for records and other AOCE catalog items (such as catalogs or dNodes) and provide lists of the record types and categories available.

SDPGetIconByType

The `SDPGetIconByType` function returns the icon suite for a specific type of record.

```
pascal OSErr SDPGetIconByType (const RString *recordType,
                               IconSelectorValue whichIcons,
                               Handle *iconSuite);
```

`recordType`
A pointer to the record type whose icon suite you want.

`whichIcons`
A selector mask that indicates which icons you want.

`iconSuite` A handle to the suite of icons you requested. The function allocates this handle. Use the Macintosh Toolbox icon utilities to handle these icons.

DESCRIPTION

If you want to display the icon for a record, you can use the `SDPGetIconByType` function to obtain one or more icons for the record. You must specify the types of icon resource you want and the type of record for which you want an icon.

To specify which icons you want for a record type, use the following mask values:

```
typedef unsigned long   IconSelectorValue;
#define svLarge1Bit     0x00000001
#define svLarge4Bit     0x00000002
#define svLarge8Bit     0x00000004
#define svSmall1Bit     0x00000100
#define svSmall4Bit     0x00000200
#define svSmall8Bit     0x00000400
#define svMini1Bit      0x00010000
#define svMini4Bit      0x00020000
#define svMini8Bit      0x00040000
#define svAllLargeData  0x000000ff
#define svAllSmallData  0x0000ff00
#define svAllMiniData   0x00ff0000
#define svAll1BitData   (svLarge1Bit | svSmall1Bit | svMini1Bit)
#define svAll4BitData   (svLarge4Bit | svSmall4Bit | svMini4Bit)
#define svAll8BitData   (svLarge8Bit | svSmall8Bit | svMini8Bit)
#define svAllAvailableData 0xffffffff
```

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetIconByType` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0400

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Icons and icon resources are described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials.*

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`. The icon utilities are described in the chapter "Icon Utilities" in *Inside Macintosh: More Macintosh Toolbox*.

SDPGetDSSpecIcon

The `SDPGetDSSpecIcon` function returns the icon for any object in the AOCE catalog system.

```
pascal OSErr SDPGetDSSpecIcon (const PackedDSSpec *packedDSSpec,
                               IconSelectorValue whichIcons,
                               Handle *iconSuite);
```

`packedDSSpec`
A pointer to the specifier of the object whose icon you want.

`whichIcons`
A selector mask that indicates which icons you want.

`iconSuite` A handle to the suite of icons you requested. The function allocates this handle. Use the Macintosh Toolbox icon utilities to handle these icons.

DESCRIPTION

If you want to display an icon for a specific attribute, a record, or a container (volume, folder, AOCE catalog, dNode, or personal catalog) in the AOCE catalog system, you can use the `SDPGetDSSpecIcon` function to obtain the icon. You must specify the types of icon resource you want and the `DSSpec` structure for the item for which you want an icon. You can use the `SDPGetPanelSelection` or `SDPGetFindPanelSelection` functions to get the `DSSpec` structure for an item the user selected.

If the object you specify does not have an icon, the system returns the default icon for objects of that type.

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`.

The selector masks for the `whichIcons` parameter are described on page 4-89.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetDSSpecIcon` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0401

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You can obtain a record type from the `DSSpec` structure returned by the `SDPGetPanelSelection` function (page 4-58) or the `SDPGetFindPanelSelection` function (page 4-82).

The routines for expanding `DSSpec` structures are described in the chapter "AOCE Utilities" in this book.

Icons and icon resources are described in the chapter "Finder Interface" in the book *Inside Macintosh: Macintosh Toolbox Essentials.*

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`. The icon utilities are described in the chapter "Icon Utilities" in *Inside Macintosh: More Macintosh Toolbox*.

SDPGetCategories

The `SDPGetCategories` function returns a list of the AOCE catalog-item categories known to the Catalog Browser.

```
pascal OSErr SDPGetCategories (
                              PackedRStringListHandle *categories,
                              PackedRStringListHandle *displayNames);
```

`categories` A pointer to a list of categories.

`displayNames`
A pointer to a list of user-readable category names.

DESCRIPTION

AOCE templates can group catalog records into categories. For example, the separate record types for LaserWriter, ImageWriter, and ImageWriter LC printers can be grouped into the category "printers." You can use the `SDPGetCategories` function to obtain a list of all the categories currently available.

SPECIAL CONSIDERATIONS

The size of the structures referenced by the handles returned by this function may be larger than a packed pathname.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetCategories` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0402

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetCategoryTypes` function (described next) to list all the types of items available within a specific category.

The `PackedRStringListHandle` structure is described in "RString List" on page 4-23.

SDPGetCategoryTypes

The `SDPGetCategoryTypes` function returns a list of the types of items within a catalog-item category known to the Catalog Browser.

```
pascal OSErr SDPGetCategoryTypes (const RString *category,
                                  PackedRStringListHandle *types);
```

`category`	The catalog-item category for which you want a list of types.
`types`	A pointer to a list of catalog-item types included in the category you specified.

DESCRIPTION

After you use the `SDPGetCategories` function to obtain a list of all the catalog-item categories currently available, you can call the `SDPGetCategoryTypes` function to list all the types of records included in a specific category. You can use this information, for example, to allow the user to choose which record types to include in the Catalog-Browsing panel or the Find panel.

Note
Before you pass the list of record types to the `SDPNewPanel` or `SDPGetNewPanel` function, you must add a record type of "DNode" (the value `kDNodeRecTypeNum`) to the types list to allow the panel to show aliases to dNodes. ◆

SPECIAL CONSIDERATIONS

The size of the structure referenced by the handle returned by this function may be larger than a packed pathname.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetCategoryTypes` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0403

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetCategories` function (page 4-91) to list all the record-type categories available.

The `PackedRStringListHandle` structure is described in "RString List" on page 4-23.

You can restrict the Catalog-Browsing panel and the Find panel to display records of specific types. For more information on this feature, see the descriptions of the `SDPNewPanel` (page 4-30), `SDPGetNewPanel` (page 4-34), and `SDPNewFindPanel` (page 4-61) functions.

Application-Defined Functions

This section describes the callback routines that you may provide for the Standard Catalog Package. Your `MyPanelBusyProc` and `MyFindPanelBusyProc` routines enable you to provide feedback to the user when the Catalog-Browsing panel and Find panel are busy.

MyPanelBusyProc

You can install a panel-busy callback routine to tell the user that the Catalog-Browsing panel is busy.

```
pascal void MyPanelBusyProc (SDPPanelHandle panel,
                             Boolean busy);
```

`panel` A handle to the panel for which you installed this routine.

`busy` A Boolean value indicating whether the panel is busy.

DESCRIPTION

You can use the `SDPInstallPanelBusyProc` function to install a panel-busy callback routine that the panel calls whenever the Catalog-Browsing panel is busy. You can use your panel-busy routine, for example, to display the Standard Catalog Package's spinning arrow icon while the panel is busy. The panel calls your routine one last time with the `busy` parameter set to `false` when the panel is no longer busy; you can then take down the spinning cursor.

You can use the following constants to obtain the icon IDs of the spinning arrows:

```
#define kFirstSpinnerIcon  -16745

#define kLastSpinnerIcon   -16738
```

SEE ALSO

Use the `SDPInstallPanelBusyProc` function (page 4-35) to install and remove panel-busy callback routines.

MyFindPanelBusyProc

You can install a Find-panel-busy callback routine that tells the user that the Find panel is busy.

```
pascal void MyFindPanelBusyProc (SDPFindPanelHandle findPanel,
                                 Boolean busy);
```

`findPanel` A handle to the Find panel for which you installed this routine.

`busy` A Boolean value indicating whether the Find panel is busy.

DESCRIPTION

You can use the `SDPInstallFindPanelBusyProc` function to install a Find-panel-busy callback routine that the Find panel calls whenever the panel is busy. You can use your callback routine, for example, to display the Standard Catalog Package's spinning arrow icon while the Find panel is busy. The Find panel calls your routine one last time with the `busy` parameter set to `false` when the Find panel is no longer busy; you can then take down the spinning cursor.

You can use the following constants to obtain the icon IDs of the spinning arrows:

```
#define kFirstSpinnerIcon  -16745

#define kLastSpinnerIcon   -16738
```

SEE ALSO

Use the `SDPInstallFindPanelBusyProc` function (page 4-65) to install and remove Find-panel-busy callback routines.

Summary of the Standard Catalog Package

C Summary

Constants and Data Types

```
#define gestaltSDPStandardDirectoryVersion   'sdvr'
#define gestaltSDPFindVersion                'dfnd'
#define gestaltSDPPromptVersion              'prpv'

/* selector mask values */
typedef unsigned long IconSelectorValue;
#define svLarge1Bit        0x00000001
#define svLarge4Bit        0x00000002
#define svLarge8Bit        0x00000004
#define svSmall1Bit        0x00000100
#define svSmall4Bit        0x00000200
#define svSmall8Bit        0x00000400
#define svMini1Bit         0x00010000
#define svMini4Bit         0x00020000
#define svMini8Bit         0x00040000
#define svAllLargeData     0x000000ff
#define svAllSmallData     0x0000ff00
#define svAllMiniData      0x00ff0000
#define svAll1BitData      (svLarge1Bit | svSmall1Bit | svMini1Bit)
#define svAll4BitData      (svLarge4Bit | svSmall4Bit | svMini4Bit)
#define svAll8BitData      (svLarge8Bit | svSmall8Bit | svMini8Bit)
#define svAllAvailableData    0xffffffff

/* generic icon suites */
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericDirectoryIconResource            -16721
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericLockedDirectoryIconResource      -16716
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericRecordIconResource               -16722
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
```

```
#define genericAttributeIconResource            -16723
/* icl8, icl4, ICN#, ics#, ics4, ics8 */
#define genericTemplateIconResource             -16746

/* standard icon suites */
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define directoryFolderIconResource             -16720
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define lockedDirectoryFolderIconResource       -16719
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define personalDirectoryIconResource           -16718
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define directoriesIconResource                 -16717
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define preferredPersonalDirectoryIconResource  -16724

/* icon IDs for spinning-arrow cursor */
#define kFirstSpinnerIcon                       -16745
#define kLastSpinnerIcon                        -16738

/* resource types */
#define kSDPPanelResourceType       'panl'
#define kSDPFindPanelResourceType   'find'

/* standard FindPanel resource */
#define kStandardFindLayout   -16700

enum {
   kSDPGuestBit,
   kSDPSpecificIdentityBit,
   kSDPLocalIdentityBit
};
/* values of SDPIdentityKind */
enum {
   kSDPGuestMask = 1<<kSDPGuestBit,
   kSDPSpecificIdentityMask = 1<<kSDPSpecificIdentityBit,
   kSDPLocalIdentityMask = 1<<kSDPLocalIdentityBit
};
typedef unsigned short SDPIdentityKind;

enum {
   kSDPSuggestionOnly,
   kSDPRestrictToDirectory,
```

```
   kSDPRestrictToRecord
};
typedef unsigned short SDPLoginFilterKind;

/* values of SDPSelectionState */
enum {
   kSDPNothingSelected,
   kSDPLockedContainerSelected,
   kSDPContainerSelected,
   kSDPRecordSelected,
   kSDPRecordAliasSelected,
   kSDPContainerAliasSelected
};
typedef unsigned short SDPSelectionState;

/* values of SDPPanelState */
enum {
   kSDPProcessed,
   kSDPSelectedAnItem,
   kSDPChangedSelection
};
typedef unsigned short SDPPanelState;

enum {
   kSDPItemIsSelectedBit,
   kSDPFindTextExistsBit
};
/* values of SDPFindPanelState */
enum {
   kSDPItemIsSelectedMask = 1<<kSDPItemIsSelectedBit,
   kSDPFindTextExistsMask = 1<<kSDPFindTextExistsBit
};
typedef unsigned short SDPFindPanelState;

/* values of SDPFindPanelFocus */
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};
typedef unsigned short SDPFindPanelFocus;
```

```
/* values of SDPFindPanelResult */
enum {
   kSDPSelectedAFindItem,
   kSDPFindSelectionChanged,
   kSDPFindCompleted,
   kSDPTextStateChanged,
   kSDPFocusChanged,
   kSDPSelectionAndFocusChanged,
   kSDPMenuChanged,
   kSDPSelectionAndMenuChanged,
   kSDPProcessedFind
};
typedef unsigned short SDPFindPanelResult;

/* Your application may read any of the fields in an SDPPanelRecord, but it
should use the appropriate routines to make changes to the records with the
exception of the refCon field, which your application may read or write at
will. Private information follows the SDPPanelRecord, so the handle must not
be resized. */

struct SDPPanelRecord {
   Rect              bounds;        /* rectangle around panel */
   Boolean           visible;       /* Is panel visible? */
   Boolean           enabled;       /* Is panel enabled? */
   Boolean           focused;       /* Is focus rectangle around panel? */
   Byte              padByte;       /* reserved */
   AuthIdentity      identity;      /* auth ID of caller of panel */
   long              refCon;        /* for your use */
   Rect              listRect;      /* rectangle around scrolling list */
   Rect              popupRect;     /* rectangle around pop-up menu */
   short             numberOfRows;  /* number of rows in scrolling list */
   short             rowHeight;     /* height of each row in list (points) */
   Boolean           pabMode;       /* Is panel in personal catalog mode? */
};

typedef struct SDPPanelRecord SDPPanelRecord;
typedef SDPPanelRecord *SDPPanelPtr, **SDPPanelHandle;

struct SDPFindPanelRecord {
   Point             upperLeft;                /* reserved */
   Boolean           visible;                  /* reserved */
   Boolean           enabled;                  /* reserved */
   Boolean           nowFinding;               /* reserved */
   Byte              padByte;                  /* reserved */
```

```
   SDPFindPanelFocus currentFocus;              /* reserved */
   AuthIdentity      identity;                  /* reserved */
   short             simultaneousSearchCount;   /* reserved */
   long              refCon;                    /* for your use */
};

typedef struct SDPFindPanelRecord SDPFindPanelRecord;
typedef SDPFindPanelRecord *SDPFindPanelPtr, **SDPFindPanelHandle;

/* pointers to functions for application-defined callback routines */

typedef pascal void (*PanelBusyProc) (SDPPanelHandle Panel,
                                      Boolean busy);

typedef pascal void (*FindPanelBusyProc) (SDPFindPanelHandle findPanel,
                                          Boolean busy);
```

Standard Catalog Package Functions

Authenticating a User

```
pascal OSErr SDPPromptForID (AuthIdentity *id,
                             ConstStr255Param guestPrompt,
                             ConstStr255Param specificIDPrompt,
                             ConstStr255Param localIDPrompt,
                             const RString *recordType,
                             SDPIdentityKind permittedKinds,
                             SDPIdentityKind *selectedKind,
                             const RecordID *loginFilter,
                             SDPLoginFilterKind filterKind);
```

Sorting a Personal Catalog

```
pascal OSErr SDPRepairPersonalDirectory
                            (FSSpec *pd, Boolean showProgress);
```

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

```
pascal OSErr SDPNewPanel    (SDPPanelHandle *newPanel,
                             WindowPtr window,
                             const Rect *bounds,
                             Boolean visible,
                             Boolean enabled,
                             const PackedRLI *initialRLI,
                             const RStringPtr *typesList,
                             unsigned long typeCount,
```

```
                              AuthIdentity identity,
                              DirEnumChoices enumFlags,
                              DirMatchWith matchTypeHow,
                              long refCon);
pascal OSErr SDPGetNewPanel (SDPPanelHandle *newPanel,
                              short resourceID,
                              WindowPtr window,
                              const PackedRLI *initialRLI,
                              AuthIdentity identity);
pascal OSErr SDPInstallPanelBusyProc
                             (SDPPanelHandle panel,
                              PanelBusyProc busyProc);
pascal OSErr SDPSetPanelBalloonHelp
                             (SDPPanelHandle panel,
                              short balloonHelpID);
pascal OSErr SDPSetIdentity (SDPPanelHandle panel,
                              AuthIdentity identity);
pascal OSErr SDPSetPath      (SDPPanelHandle panel,
                              const PackedRLI *prli);
pascal OSErr SDPGetPathLength
                             (SDPPanelHandle panel,
                              unsigned short *pathNameLength);
pascal OSErr SDPGetPath      (SDPPanelHandle panel,
                              PackedRLI *prli,
                              short *dsRefNum);
pascal OSErr SDPSelectString
                             (SDPPanelHandle panel,
                              const RString *name);
pascal OSErr SDPHidePanel    (SDPPanelHandle panel);
pascal OSErr SDPShowPanel    (SDPPanelHandle panel);
pascal OSErr SDPEnablePanel (SDPPanelHandle panel,
                              Boolean enable);
pascal OSErr SDPSetFocus     (SDPPanelHandle panel,
                              Boolean focus);
pascal OSErr SDPUpdatePanel (SDPPanelHandle panel,
                              RgnHandle theRgn);
pascal OSErr SDPMovePanel    (SDPPanelHandle panel,
                              short h,
                              short v);
pascal OSErr SDPSizePanel    (SDPPanelHandle panel,
                              short width,
                              short height);
```

```
pascal OSErr SDPDisposePanel
                            (SDPPanelHandle panel);
```

Handling Events in a Catalog-Browsing Panel

```
pascal OSErr SDPPanelEvent  (SDPPanelHandle panel,
                             const EventRecord *theEvent,
                             SDPPanelState *whatHappened);
pascal OSErr SDPGetPanelSelectionState
                            (SDPPanelHandle panel,
                             SDPSelectionState *itsState);
pascal OSErr SDPGetPanelSelectionSize
                            (SDPPanelHandle panel,
                             unsigned short *dsSpecSize);
pascal OSErr SDPGetPanelSelection
                            (SDPPanelHandle panel,
                             PackedDSSpec *selection);
pascal OSErr SDPOpenSelectedItem
                            (SDPPanelHandle panel,
                             SDPPanelState *whatHappened);
```

Creating, Displaying, and Disposing of a Find Panel

```
pascal OSErr SDPNewFindPanel
                            (SDPFindPanelHandle *newPanel,
                             WindowPtr window,
                             Point upperLeft,
                             short layoutResourceID,
                             Boolean visible,
                             Boolean enabled,
                             const RStringPtr *typesList,
                             long typeCount,
                             DirMatchWith matchTypeHow,
                             AuthIdentity identity,
                             short simultaneousSearchCount,
                             SDPFindPanelFocus initialFocus,
                             long refCon);
pascal OSErr SDPInstallFindPanelBusyProc
                            (SDPFindPanelHandle findPanel,
                             FindPanelBusyProc busyProc);
pascal OSErr SDPSetFindPanelBalloonHelp
                            (SDPFindPanelHandle findPanel,
                             short balloonHelpID);
pascal OSErr SDPHideFindPanel
                            (SDPFindPanelHandle findPanel);
```

```
pascal OSErr SDPShowFindPanel
                            (SDPFindPanelHandle findPanel);
pascal OSErr SDPEnableFindPanel
                            (SDPFindPanelHandle findPanel,
                             Boolean enabled);
pascal OSErr SDPSetFindPanelFocus
                            (SDPFindPanelHandle findPanel,
                             SDPFindPanelFocus newFocus);
pascal OSErr SDPSetFindIdentity
                            (SDPFindPanelHandle findPanel,
                             AuthIdentity identity)
pascal OSErr SDPUpdateFindPanel
                            (SDPFindPanelHandle findPanel,
                             RgnHandle theRgn);
pascal OSErr SDPMoveFindPanel
                            (SDPFindPanelHandle findPanel,
                             short h,
                             short v);
pascal OSErr SDPDisposeFindPanel
                            (SDPFindPanelHandle findPanel);
```

Handling Events in a Find Panel

```
pascal OSErr SDPFindPanelEvent
                            (SDPFindPanelHandle findPanel,
                             const EventRecord *event,
                             SDPFindPanelResult *whatHappened);
pascal OSErr SDPGetFindPanelState
                            (SDPFindPanelHandle findPanel,
                             SDPFindPanelState *itsState);
pascal OSErr SDPGetFindPanelSelectionSize
                            (SDPFindPanelHandle findPanel,
                             unsigned short *size);
pascal OSErr SDPGetFindPanelSelection
                            (SDPFindPanelHandle findPanel,
                             PackedDSSpec *selection);
pascal OSErr SDPStartFind   (SDPFindPanelHandle findPanel);
pascal OSErr SDPStopFind    (SDPFindPanelHandle findPanel);
```

Resolving Aliases

```
pascal OSErr SDPResolveAliasFile
                            (FSSpecPtr fileSpec,
                             PackedDSSpecHandle resolvedDSSpec,
                             AuthIdentity identity,
                             Boolean mayPromptUser);
pascal OSErr SDPResolveAliasDSSpec
                            (PackedDSSpecHandle theAliasDSSpec,
                             AuthIdentity identity,
                             Boolean mayPromptUser);
```

Obtaining Icons and Lists of Catalog-Item Categories and Types

```
pascal OSErr SDPGetIconByType
                            (const RString *recordType,
                             IconSelectorValue whichIcons,
                             Handle *iconSuite);
pascal OSErr SDPGetDSSpecIcon
                            (const PackedDSSpec *packedDSSpec,
                             IconSelectorValue whichIcons,
                             Handle *iconSuite);
pascal OSErr SDPGetCategories
                            (PackedRStringListHandle *categories,
                             PackedRStringListHandle *displayNames);
pascal OSErr SDPGetCategoryTypes
                            (const RString *category,
                             PackedRStringListHandle *types);
```

Application-Defined Functions

```
pascal void MyPanelBusyProc (SDPPanelHandle panel,
                             Boolean busy);
pascal void MyFindPanelBusyProc
                            (SDPFindPanelHandle findPanel,
                             Boolean busy);
```

Pascal Summary

Constants

```
CONST
gestaltSDPStandardDirectoryVersion= 'sdvr';
gestaltSDPFindVersion= 'dfnd';
gestaltSDPPromptVersion= 'prpv';

{ selector mask values }
svLarge1Bit= $00000001;
svLarge4Bit= $00000002;
svLarge8Bit= $00000004;
svSmall1Bit= $00000100;
svSmall4Bit= $00000200;
svSmall8Bit= $00000400;
svMini1Bit= $00010000;
svMini4Bit= $00020000;
svMini8Bit= $00040000;
svAllLargeData= $000000ff;
svAllSmallData= $0000ff00;
svAllMiniData= $00ff0000;
svAll1BitData= (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData= (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData= (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableData= $ffffffff;

{ generic icon suites }
genericDirectoryIconResource= -16721;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericLockedDirectoryIconResource= -16716;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericRecordIconResource= -16722;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericAttributeIconResource= -16723;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericTemplateIconResource= -16746;
{ icl8, icl4, ICN#, ics#, ics4, ics8 }

{ standard icon suites }
directoryFolderIconResource= -16720;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
lockedDirectoryFolderIconResource= -16719;
```

```
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
personalDirectoryIconResource= -16718;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
directoriesIconResource= -16717;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
preferredPersonalDirectoryIconResource= -16724;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }

{ icon IDs for spinning-arrow cursor }
kFirstSpinnerIcon = -16745;
kLastSpinnerIcon  = -16738;

{ resource types }
kSDPPanelResourceType= 'panl';
kSDPFindPanelResourceType= 'find';

{ standard FindPanel resource }
kStandardFindLayout= -16700;

kSDPGuestBit = 0;
kSDPSpecificIdentityBit = 1;
kSDPLocalIdentityBit = 2;

{ values of SDPIdentityKind }
kSDPGuestMask              = $0001; { 1<<kSDPGuestBit }
kSDPSpecificIdentityMask   = $0002; { 1<<kSDPSpecificIdentityBit }
kSDPLocalIdentityMask      = $0004; { 1<<kSDPLocalIdentityBit }

kSDPSuggestionOnly = 0;
kSDPRestrictToDirectory = 1;
kSDPRestrictToRecord = 2;

{ values of SDPSelectionState }
kSDPNothingSelected = 0;
kSDPLockedContainerSelected = 1;
kSDPContainerSelected = 2;
kSDPRecordSelected = 3;
kSDPRecordAliasSelected = 4;
kSDPContainerAliasSelected = 5;

{ values of SDPPanelState }
kSDPProcessed = 0;
kSDPSelectedAnItem = 1;
kSDPChangedSelection = 2;
```

```
kSDPItemIsSelectedBit = 0;
kSDPFindTextExistsBit = 1;

{ values of SDPFindPanelState }
kSDPItemIsSelectedMask = $0001; { 1<<kSDPItemIsSelectedBit }
kSDPFindTextExistsMask = $0002; { 1<<kSDPFindTextExistsBit }

{ values of SDPFindPanelFocus }
kSDPFindPanelNoFocus = 0;
kSDPFindPanelListHasFocus = 1;
kSDPFindPanelTextHasFocus = 2;

{ values of SDPFindPanelResult }
kSDPSelectedAFindItem = 0;
kSDPFindSelectionChanged = 1;
kSDPFindCompleted = 2;
kSDPTextStateChanged = 3;
kSDPFocusChanged = 4;
kSDPSelectionAndFocusChanged = 5;
kSDPMenuChanged = 6;
kSDPSelectionAndMenuChanged = 7;
kSDPProcessedFind = 8;
```

Data Types

```
TYPE
IconSelectorValue = LONGINT;

SDPMatchWith = INTEGER;

SDPIdentityKind = INTEGER;

SDPLoginFilterKind = INTEGER;

SDPSelectionState = INTEGER;

SDPPanelState = INTEGER;

SDPFindPanelState = INTEGER;

SDPFindPanelFocus = INTEGER;

SDPFindPanelResult = INTEGER;
```

```
{ Your application may read any of the fields in an SDPPanelRecord, but it
should use the appropriate routines to make changes to the records with the
exception of the refCon field, which your application may read or write at
will. Private information follows the SDPPanelRecord, so the handle must not
be resized. }

SDPPanelRecord = RECORD
   bounds: Rect;
   visible: BOOLEAN;
   enabled: BOOLEAN;
   focused: BOOLEAN;
   {padByte: Byte;}
   identity: AuthIdentity;
   refCon: LONGINT;
   listRect: Rect;
   popupRect: Rect;
   numberOfRows: INTEGER;
   rowHeight: INTEGER;
   pabMode: BOOLEAN;
   END;

SDPPanelPtr = ^SDPPanelRecord;
SDPPanelHandle = ^SDPPanelPtr;

SDPFindPanelRecord = RECORD
   upperLeft: Point;
   visible: BOOLEAN;
   enabled: BOOLEAN;
   nowFinding: BOOLEAN;
   {padByte: Byte;}
   currentFocus: SDPFindPanelFocus;
   identity: AuthIdentity;
   simultaneousSearchCount: INTEGER;
   refCon: LONGINT;
   END;

SDPFindPanelPtr = ^SDPFindPanelRecord;
SDPFindPanelHandle = ^SDPFindPanelPtr;

PackedRStringListHandle = ^PackedPathNamePtr;
```

```
PackedDSSpecHandle = ^PackedDSSpecPtr;

PanelBusyProc = ProcPtr;
   { PROCEDURE PanelBusyProc(Panel: SDPPanelHandle; busy: BOOLEAN); }

FindPanelBusyProc = ProcPtr;
   {PROCEDURE FindPanelBusyProc(findPanel: SDPFindPanelHandle;
                                                busy: BOOLEAN); }
```

Standard Catalog Package Functions

Authenticating a User

```
FUNCTION SDPPromptForID     (VAR id: AuthIdentity;
                             guestPrompt: StringPtr;
                             specificIDPrompt: StringPtr;
                             localIDPrompt: StringPtr;
                             recordType: RStringPtr;
                             permittedKinds: SDPIdentityKind;
                             VAR selectedKind: SDPIdentityKind;
                             loginFilter: RecordIDPtr;
                             filterKind: SDPLoginFilterKind): OSErr;
```

Sorting a Personal Catalog

```
FUNCTION SDPRepairPersonalDirectory
                            (pd: FSSpecPtr; showProgress: BOOLEAN): OSErr;
```

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

```
FUNCTION SDPNewPanel        (VAR newPanel: SDPPanelHandle;
                             window: WindowPtr;
                             bounds: Rect;
                             visible: BOOLEAN;
                             enabled: BOOLEAN;
                             initialRLI: PackedRLIPtr;
                             typesList: RStringHandle;
                             typeCount: LONGINT;
                             identity: AuthIdentity;
                             enumFlags: DirEnumChoices;
                             matchTypeHow: SDPMatchWith;
                             refCon: LONGINT): OSErr;
```

```
FUNCTION SDPGetNewPanel     (VAR newPanel: SDPPanelHandle;
                             resourceID: INTEGER;
                             window: WindowPtr;
                             initialRLI: PackedRLIPtr;
                             identity: AuthIdentity): OSErr;
FUNCTION SDPInstallPanelBusyProc
                            (panel: SDPPanelHandle;
                             busyProc: PanelBusyProc): OSErr;
FUNCTION SDPSetPanelBalloonHelp
                            (panel: SDPPanelHandle;
                             balloonHelpID: INTEGER): OSErr;
FUNCTION SDPSetIdentity     (panel: SDPPanelHandle; identity:
                             AuthIdentity): OSErr;
FUNCTION SDPSetPath         (panel: SDPPanelHandle; prli: PackedRLIPtr):
                             OSErr;
FUNCTION SDPGetPathLength   (panel: SDPPanelHandle;
                             VAR pathNameLength: INTEGER): OSErr;
FUNCTION SDPGetPath         (panel: SDPPanelHandle; prli: PackedRLIPtr;
                             VAR dsRefNum: INTEGER): OSErr;
FUNCTION SDPSelectString    (panel: SDPPanelHandle; name: RStringPtr):
                             OSErr;
FUNCTION SDPHidePanel       (panel: SDPPanelHandle): OSErr;
FUNCTION SDPShowPanel       (panel: SDPPanelHandle): OSErr;
FUNCTION SDPEnablePanel     (panel: SDPPanelHandle; enable: BOOLEAN): OSErr;
FUNCTION SDPSetFocus        (panel: SDPPanelHandle; focus: BOOLEAN): OSErr;
FUNCTION SDPUpdatePanel     (panel: SDPPanelHandle; theRgn: RgnHandle):
                             OSErr;
FUNCTION SDPMovePanel       (panel: SDPPanelHandle; h: INTEGER; v:
                             INTEGER): OSErr;
FUNCTION SDPSizePanel       (panel: SDPPanelHandle; width: INTEGER;
                             height: INTEGER): OSErr;
FUNCTION SDPDisposePanel    (panel: SDPPanelHandle): OSErr;
```

Handling Events in a Catalog-Browsing Panel

```
FUNCTION SDPPanelEvent      (panel: SDPPanelHandle;
                             theEvent: EventRecord;
                             VAR whatHappened: SDPPanelState): OSErr;
FUNCTION SDPGetPanelSelectionState
                            (panel: SDPPanelHandle;
                             VAR itsState: SDPSelectionState): OSErr;
FUNCTION SDPGetPanelSelectionSize
                            (panel: SDPPanelHandle;
                             VAR dsSpecSize: INTEGER): OSErr;
FUNCTION SDPGetPanelSelection
                            (panel: SDPPanelHandle;
                             selection: PackedDSSpecPtr): OSErr;
FUNCTION SDPOpenSelectedItem
                            (panel: SDPPanelHandle;
                             VAR whatHappened: SDPPanelState): OSErr;
```

Creating, Displaying, and Disposing of a Find Panel

```
FUNCTION SDPNewFindPanel    (VAR newPanel: SDPFindPanelHandle;
                             window: WindowPtr;
                             upperLeft: Point;
                             layoutResourceID: INTEGER;
                             visible: BOOLEAN;
                             enabled: BOOLEAN;
                             typesList: RStringHandle;
                             typeCount: LONGINT;
                             matchTypeHow: SDPMatchWith;
                             identity: AuthIdentity;
                             simultaneousSearchCount: INTEGER;
                             initialFocus: SDPFindPanelFocus;
                             refCon: LONGINT): OSErr;
FUNCTION SDPInstallFindPanelBusyProc
                            (findPanel: SDPFindPanelHandle; busyProc:
                             FindPanelBusyProc): OSErr;
FUNCTION SDPSetFindPanelBalloonHelp
                            (findPanel: SDPFindPanelHandle;
                             balloonHelpID: INTEGER): OSErr;
FUNCTION SDPHideFindPanel   (findPanel: SDPFindPanelHandle): OSErr;
FUNCTION SDPShowFindPanel   (findPanel: SDPFindPanelHandle): OSErr;
```

```
FUNCTION SDPEnableFindPanel (findPanel: SDPFindPanelHandle;
                             enabled: BOOLEAN): OSErr;
FUNCTION SDPSetFindPanelFocus
                            (findPanel: SDPFindPanelHandle;
                             newFocus: SDPFindPanelFocus): OSErr;
FUNCTION SDPSetFindIdentity
                            (findPanel: SDPFindPanelHandle;
                             identity: AuthIdentity): OSErr;
FUNCTION SDPUpdateFindPanel
                            (findPanel: SDPFindPanelHandle;
                             theRgn: RgnHandle): OSErr;
FUNCTION SDPMoveFindPanel   (findPanel: SDPFindPanelHandle; h: INTEGER;
                             v: INTEGER): OSErr;
FUNCTION SDPDisposeFindPanel
                            (findPanel: SDPFindPanelHandle): OSErr;
```

Handling Events in a Find Panel

```
FUNCTION SDPFindPanelEvent  (findPanel: SDPFindPanelHandle;
                             event: EventRecord;
                             VAR whatHappened: SDPFindPanelResult): OSErr;
FUNCTION SDPGetFindPanelState
                            (findPanel: SDPFindPanelHandle;
                             VAR itsState: SDPFindPanelState): OSErr;
FUNCTION SDPGetFindPanelSelectionSize
                            (findPanel: SDPFindPanelHandle;
                             VAR size: INTEGER): OSErr;
FUNCTION SDPGetFindPanelSelection
                            (findPanel: SDPFindPanelHandle;
                             selection: PackedDSSpecPtr): OSErr;
FUNCTION SDPStartFind       (findPanel: SDPFindPanelHandle): OSErr;
FUNCTION SDPStopFind        (findPanel: SDPFindPanelHandle): OSErr;
```

Resolving Aliases

```
FUNCTION SDPResolveAliasFile
                            (fileSpec: FSSpecPtr;
                             resolvedDSSpec: PackedDSSpecHandle;
                             identity: AuthIdentity;
                             mayPromptUser: BOOLEAN): OSErr;
FUNCTION SDPResolveAliasDSSpec
                            (theAliasDSSpec: PackedDSSpecHandle;
                             identity: AuthIdentity;
                             mayPromptUser: BOOLEAN): OSErr;
```

Obtaining Icons and Lists of Catalog-Item Categories and Types

```
FUNCTION SDPGetIconByType   (recordType: RStringPtr;
                             whichIcons: IconSelectorValue;
                             VAR iconSuite: Handle): OSErr;
FUNCTION SDPGetDSSpecIcon   (packedDSSpec: PackedDSSpecPtr;
                             whichIcons: IconSelectorValue;
                             VAR iconSuite: Handle): OSErr;
FUNCTION SDPGetCategories   (VAR catagories: PackedRStringListHandle;
                             VAR displayNames: PackedRStringListHandle):
                             OSErr;
FUNCTION SDPGetCategoryTypes
                            (category: RStringPtr;
                             VAR types: PackedRStringListHandle): OSErr;
```

Application-Defined Functions

```
PROCEDURE MyPanelBusyProc   (panel: SDPPanelHandle; busy: BOOLEAN);
PROCEDURE MyFindPanelBusyProc
                            (findPanel: SDPFindPanelHandle; busy: BOOLEAN);
```

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

$AA5D

Selector	Count	Function
$0388	$0010	SDPPromptForID
$1A2C	$0003	SDPRepairPersonalDirectory
$0064	$0015	SDPNewPanel
$0065	$0009	SDPGetNewPanel
$0079	$0004	SDPInstallPanelBusyProc
$0078	$0003	SDPSetPanelBalloonHelp
$0073	$0004	SDPSetIdentity
$0070	$0004	SDPSetPath
$0075	$0004	SDPGetPathLength
$0076	$0006	SDPGetPath
$0074	$0004	SDPSelectString
$0067	$0002	SDPHidePanel
$0068	$0002	SDPShowPanel
$0069	$0003	SDPEnablePanel
$0077	$0003	SDPSetFocus
$006A	$0004	SDPUpdatePanel
$006B	$0004	SDPMovePanel
$006C	$0004	SDPSizePanel
$0066	$0002	SDPDisposePanel
$0071	$0006	SDPPanelEvent
$006E	$0004	SDPGetPanelSelectionState
$0072	$0004	SDPGetPanelSelectionSize
$006F	$0004	SDPGetPanelSelection
$006D	$0004	SDPOpenSelectedItem
$08FC	$0014	SDPNewFindPanel
$090C	$0004	SDPInstallFindPanelBusyProc
$090A	$0003	SDPSetFindPanelBalloonHelp
$0903	$0002	SDPHideFindPanel
$0902	$0002	SDPShowFindPanel
$0905	$0003	SDPEnableFindPanel

Selector	Count	Function
$0906	$0003	SDPSetFindPanelFocus
$090B	$0004	SDPSetFindIdentity
$0901	$0004	SDPUpdateFindPanel
$0904	$0004	SDPMoveFindPanel
$08FD	$0002	SDPDisposeFindPanel
$0900	$0006	SDPFindPanelEvent
$0907	$0004	SDPGetFindPanelState
$0908	$0004	SDPGetFindPanelSelectionSize
$0909	$0004	SDPGetFindPanelSelection
$08FE	$0002	SDPStartFind
$08FF	$0002	SDPStopFind
$0E74	$0007	SDPResolveAliasFile
$0E75	$0005	SDPResolveAliasDSSpec

$AA5C

Selector	Function
$0400	SDPGetIconByType
$0401	SDPGetDSSpecIcon
$0402	SDPGetCategories
$0403	SDPGetCategoryTypes

Result Codes

The allocated range of result codes for the Standard Catalog Package is –1950 through –1969. Routines may also return standard Macintosh result codes such as `noErr` 0 (no error) and `fnfErr` –43 (file not found).

kSDPNoSearchText	–1950	No text is in Find panel text field
kSDPTooManyLoginAttempts	–1951	User tried more than 3 incorrect passwords
kSDPNoSelection	–1952	No selection is in Catalog-Browsing panel
kSDPPersonalDirectoryRepairFailed	–1953	Cannot sort personal catalog